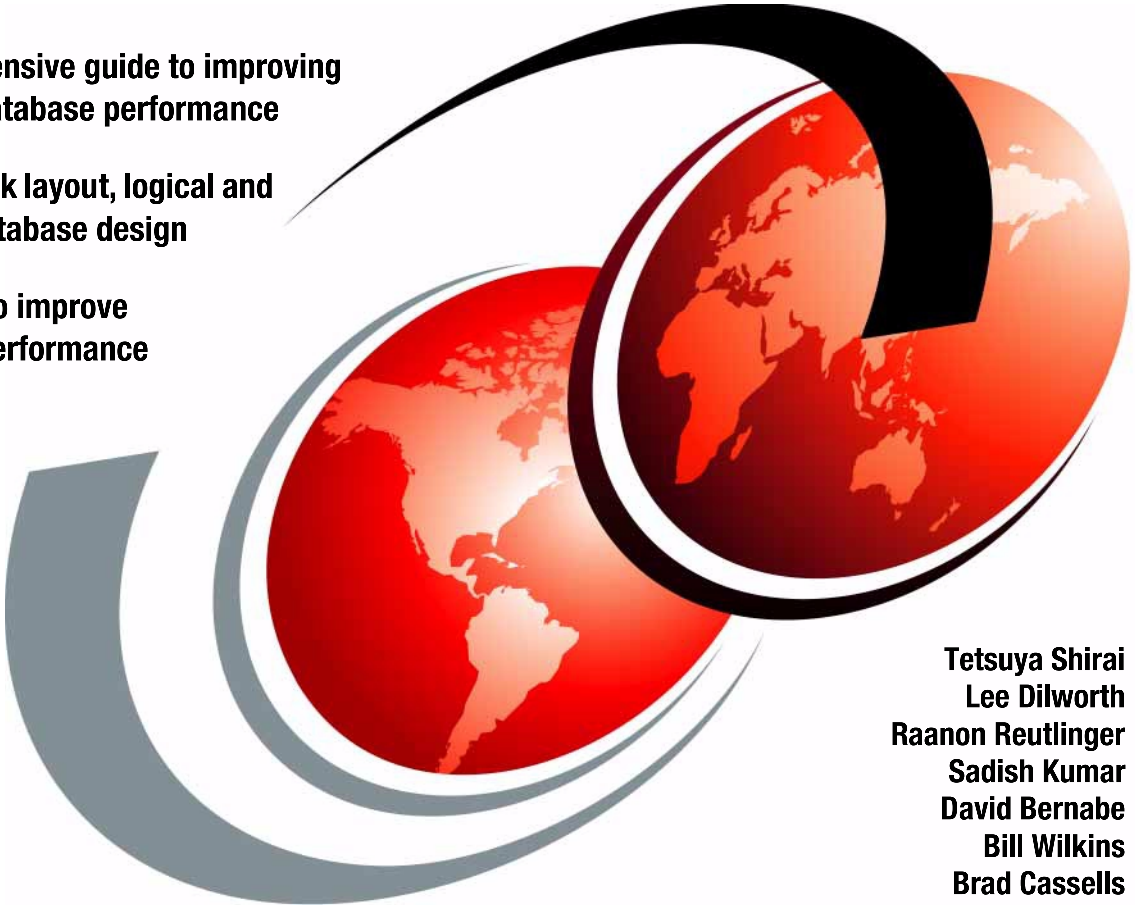
IBM

DB2 UDB V7.1 Performance Tuning Guide

A comprehensive guide to improving DB2 UDB database performance

Efficient disk layout, logical and physical database design

Many tips to improve database performance



Tetsuya Shirai
Lee Dilworth
Raanon Reutlinger
Sadish Kumar
David Bernabe
Bill Wilkins
Brad Cassells

Redbooks

ibm.com/redbooks

SG24-6012-00

International Technical Support Organization

DB2 UDB V7.1 Performance Tuning Guide

December 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special notices” on page 381.

First Edition (December 2000)

This edition applies to Version 7, Release 1 of IBM DB2 Universal Database, Program Number 5648-D48 for use with the AIX V4.3.3 operating system.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

DB2 Universal Database (DB2 UDB) is IBM's relational database management system that is at the core of many business-critical systems. It is fully scalable from a single processor to symmetric multiple processors and to massively parallel clusters, and features multimedia capabilities with image, audio, video, text, spatial and other object relational support. Because of its compelling features, power, and flexibility, many customers from small businesses to large financial institutions have chosen DB2 UDB.

DB2 UDB environments range from stand-alone systems to complex combinations of database servers and clients running on multiple platforms. In any type of system, the common key for successful applications is performance.

Particularly in a large and complex environment, performance analysis and tuning are difficult tasks and require a good understanding of the environment as well as the tools that you will use.

This IBM Redbook will provide you with guidelines for system design, database design, and application design with DB2 UDB for AIX Version 7.1. We will also discuss the methods that are available for performance analysis and tuning.

Prior to reading this book, you need to have some knowledge of database environments, as well as a basic understanding of activities that are typically performed in a database environment.

This book was written from the AIX operating system perspective, and some tools and commands discussed in this book may not be available on other operating systems. However, the hints and methodologies described in this book can be applicable for database systems which use DB2 UDB on other operating systems.

This publication does not cover partitioned databases, which you can create using DB2 UDB Enterprise-Extended Edition (EEE). Please see the *DB2 UDB Administration Guide - Planning,* SC09-2946, *the DB2 UDB Administration Guide - Implementation,* SC09-2944, and the *DB2 UDB Administration Guide - Performance,* SC09-2945 for hints and tips for partitioned databases using DB2 UDB EEE.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization San Jose Center.

Tetsuya Shirai is a Project Leader at the International Technical Support Organization (ITSO), San Jose Center. He worked in Tokyo, Japan before joining the San Jose ITSO. He has been with IBM for 8 years, working for the last 4 years with DB2 UDB products. He has worked in the area of DB2 services, teaching classes to both customers and IBMers. He has also provided consulting services to DB2 customers worldwide.

Lee Dilworth is a Software Support Specialist working within the RS/6000 Support Centre based in Basingstoke, UK. He has worked for IBM for 3 years. His main areas of expertise are DB2 UDB, AIX, and RS/6000 SP systems. He is an IBM Certified, Advanced Technical Expert and SP specialist and is currently the DB2 technical advisor within the Support Centre, working with DB2 on Solaris and Linux as well as AIX.

Raanon Reutlinger has been a Technical Sales Specialist in the IBM Software Group for the last 5 years, responsible for both pre-sale and post-sale support in Israel. He is a Data Management I/T Specialist and is a Certified Solutions Expert for DB2 Administration and DB2 Application Development. Raanon holds a degree in Computer Science and has 15 years experience in Database/Application Development projects, on the UNIX and Windows platforms, both in the US and in Israel. His areas of expertise include administration, development, monitoring, and tuning with DB2 UDB EE and EEE, as well as DataJoiner, Data Replication, and the Visual Warehouse products.

Sadish Kumar is a Software Engineer with DB2 Universal Database World Trade Support in IGSI, India. He has been with the Level 2 Engine and Application Support team for 1-1/2 years supporting customers on Intel and UNIX platforms. His areas of expertise include installation, administration, and problem determination in DB2 UDB EE environments.

David Bernabe works for the Integrated Technology Services organization in IBM Spain. He has over 6 years of experience as a Database Specialist working in application development and database administration. He has provided technical support for DB2 UDB and Data Warehouse products on UNIX and Intel platforms since he joined IBM in 1998.

Bill Wilkins is a Technical Specialist in the IBM Data Management Consulting Services group in Toronto. Prior to this he worked in DB2 performance analysis in the IBM Toronto development laboratory for 6 years, including 2 years as manager. He is an IBM Certified Advanced Technical Expert in DB2 UDB and has over 20 years of experience in the IT industry.

Brad Cassells is the Lead Writer on the DB2 UDB Administration Guide in the Information Development group of the IBM Toronto Software Development Laboratory. For the last 8 years he has worked on DB2 UDB information as team leader, planner, and writer. He is an IBM Certified Solutions Expert on DB2 UDB Database Administration. In addition, he has 5 years of DB2 for VM and VSE experience, and has worked at IBM for 17 years.

Thanks to the following people for their invaluable contributions to this project:

Berni Schiefer
Matthew Huras
Roger Zheng
Haider Rizvi
Cadambi Sriram
Hebert Pereyra
Kaarel Truuvert
Dale McInnis
Adriana Zubiri
Lily Lugomirski
Dwaine Snow
Sean McKeough
Mike Winer
Susan Visser
IBM Toronto Lab

Mikiko Satoh
IBM Yamato Lab

Nhut Bui
IBM Silicon Valley Lab

John Aschoff
IBM Storage Systems Division

Mary Comianos
Emma Jacobs
Yvonne Lyon
Deanna Polm
International Technical Support Organization, San Jose Center

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 395 to the fax number shown on the form.
- Use the online evaluation form found at **ibm.com**/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Overview

DB2 UDB environments range from stand-alone systems to complex combinations of database servers and clients running on multiple platforms. In any type of system, the common key for successful applications is performance.

When you plan, design, and build your database system, you need to understand several considerations about logical and physical database design, application design, and configuration parameters of DB2 so that your database system can meet the performance requirements of your application.

While its performance may initially be good, as time goes on, your system may need to serve more users, store more data, and process more complex queries. Consequently, the increased load level of your database server will affect its performance. This could be the time to upgrade to more powerful equipment. However, before investing in equipment, you may be able to improve performance by simply tuning the database system.

This chapter provides an overview of the tasks involved in tuning the database environment to obtain optimal performance. It also provides an overview of the architecture and processes of DB2 UDB.

1.1 Measuring system performance

Performance is the capacity of your system to produce the desired results with a minimum cost of time or resources. It can be measured through response time, throughput, and availability.

Performance is not an absolute value. The performance of an information system can be rated as better or worse, compared to a reference value. First, the reference value needs to be established according to the requirements of the information system; then the results of tuning efforts can be compared against it. Those requirements, or the service level agreement, may include the throughput of the system, limits on the response time for a percentile of transactions, or any other issues relevant to the end user.

Units of measurement are usually based on the response time of a given workload. Other units of measurement may be based on transactions per second, I/O operations, CPU use, or a combination of the above.

Avoid setting limited performance goals such as “The response time of all transactions must be less than 3 seconds”. This is not a practical goal, because your application may submit a complex query which takes 10 minutes (even though this does not happen everyday). Rather, a more reasonable goal might be “This particular query must be completed in 2 minutes”. Once you have set a measurable performance goal, monitor your system’s actual performance and compare it with that goal. Then consider what you can do to fill the gap.

1.2 Determining when system tuning will be cost-effective

Your database system is a complex data-processing environment that includes hardware resources, software components, and application programs. DB2 starts many processes that perform different functions in your database system, and it allocates the necessary memory areas, thus consuming hardware resources.

As system performance degrades, you may at first be tempted to upgrade your system with more powerful and expensive equipment. However, in the meantime, you may be able to improve the performance of your existing resources by simply tuning your operating system and databases. By carrying out a performance tuning project, you can balance your hardware resources to each part of the database system, including processes and the required memory areas.

Specific goals of tuning could include:

- Processing a larger or more demanding workload without buying new hardware.
- Obtaining faster system response times, or higher throughput, without increasing processing costs.
- Reducing processing costs without negatively affecting service to your users, and spending the money for other resources.

Other benefits are intangible; for example, greater user satisfaction and productivity resulting from faster response times. If you manage an Internet business, higher performance — including quick response time and high availability — may prevent lost business opportunities. When weighing the cost of performance tuning against its possible benefits, all of these benefits need to be considered.

1.3 Causes of performance problems

There are many possible causes of performance problems. Below we describe those most frequently encountered.

Poor application design
When you experience performance problems, in many cases these stem from poor application design and inefficient programs. The database itself may not have any problems at all. For example, SQL statements written inappropriately can degrade overall performance, even though your database is well designed and tuned. Chapter 7, "Tuning application performance" on page 229 describes tips you should take into consideration when designing and developing applications.

Poor system and database design
Poor design of your system and/or databases can be also a reason for performance problems. Inefficient disk storage layout, or failing to consider performance when designing and implementing your database, will certainly degrade performance, and can be very difficult to fix once your production system has been started. Chapter 3, "Data storage management for performance" on page 31 describes tips you should take into consideration when designing disk storage layout and databases.

System resource shortages
System resource shortages can cause bottlenecks in your database system:

- **CPU**
  Too many users, or running applications on a CPU, may cause system degradation.
- **Memory**
  Every process will use some physical memory. If you have insufficient memory, you may find that applications will fail, or your system will start thrashing.
- **Disk I/O**
  I/O performance can play an important part in a database system. Too much activity on a single disk or I/O bus may cause performance problems.
- **Network**
  Unless you are in a stand-alone environment, the network plays an important role. If the network is too slow, then this may appear to the client as a performance problem at the database server.

1.4 Deciding when to tune the system

As mentioned in the previous section, if the reason for poor performance exists in the system or database design, this will not be as easy to fix as simply tuning performance parameters of the operating system or the database manager. Therefore, you should be aware that performance tuning is not an add-on activity to be done in the production system when there is a problem. Performance needs to be a design goal. It must be taken into consideration in each step of the project life cycle:

- **Planning**
  Choose the right hardware, software, and network for your system. Take care of future growth.
- **Design**
  Choose an appropriate data model and programming interfaces.
- **Development**
  Take performance into consideration when developing application programs. Choose the right locking strategy.
- **Testing/acceptance**
  Use the same volume of data and configuration as the production system.
- **Production**
  Perform proactive performance tuning.

In this book, we discuss not only tuning the operating system and the database manager/database configuration parameters, but also various considerations associated with disk storage layout, database design, and application design.

1.5 Planning performance tuning

When you carry out a performance tuning project, the worst possible approach is to change the value of many performance tuning parameters without having any idea of what is causing the performance problem. Barring miracles, performance will only get worse, and you will never know which parameter was the cause. So, even if you are anxious for results, you will benefit from following a more methodical approach:

1. Find which queries are slow.
2. Measure the current performance and set the performance goal.
3. Monitor your system and identify where the bottleneck is.
4. Decide where you can afford to make trade-offs, and which resources can bear an additional load.
5. Change only one performance parameter to relieve the bottleneck.
6. Execute the queries again to monitor your system, and check if the performance meets your goal.

7. If the performance does not meet the goal, go back to step 3.

1.5.1 Locate problems and establish goals

When a user reports that response time is too slow, you first need to identify what the problem is. You may need to check:

- Whether this user is the only one who has experienced this problem, or whether others are also complaining about the same problem.
- Whether the poor performance was experienced only in a particular application or queries.
- Whether the problem is really one of response time. Sometimes, another problem (such as the application waiting for a response from the user) might be the cause of poor response time.

Also, you should check:

- When the user just noticed the poor performance today, or has been seeing it for a long time.
- How slow it is — ten percent slower, or ten times slower usual — for example.
- Whether someone had made significant changes on the system just before the performance problem was noticed.

To carry out a performance tuning project, it is important to establish the objective. Find which queries are slow, and set a measurable performance goal such as “The response time of this particular query should be less than 30 seconds”.

1.5.2 Identify the cause

Once you have focused on particular queries, execute those queries and monitor the system by using the monitoring tools which the operating system and DB2 provide, and identify where the bottleneck is. Chapter 5, “Monitoring tools and utilities” on page 119 presents several tools you can use.

If possible, you should execute the queries in the system where the problem has been observed. If you have to execute them in another environment, it is important to replicate the original environment as carefully as possible. For example, if the user is running in client-server mode, then you should set up a client-server environment to test the queries.

This step is important, because if you tune resources that are not the primary cause of performance problems, your efforts will have little or no effect on response time until you have relieved the major constraints. Also, the changes you introduce can make subsequent tuning work more difficult.

1.5.3 Change one performance parameter at a time

Even if you find more than one possible cause and you are sure that tuning all of them will be beneficial, you should still change only one performance tuning parameter at a time. It will be very difficult to evaluate how much benefit each change has contributed if you change more than one parameter.

The appropriate values for parameters affecting performance can best be determined by performing tests and modifying the values of the parameters until the point of diminishing returns for performance is found. If performance versus parameter values were to be graphed, the point where the curve begins to plateau or decline would indicate the point at which additional allocation provides no additional value to the application, and is therefore simply wasting memory.

DB2 provides configuration parameters to balance your hardware resources to each part of the database system, including processes and required memory areas. These configuration parameters can be divided into two groups — the database manager configuration parameters and the database configuration parameters, depending on whether their settings affect instance level, or database level.

To update the database manager configuration parameters, you can use the Control Center GUI tools (as discussed in Chapter 2, “Setting up the Control Center” on page 15). Or, you can use the following command:

```
UPDATE DBM CFG USING parameter_name value
```

To update the database configuration parameters, you can use the Control Center GUI tools, or use the following command:

```
UPDATE DB CFG FOR dbname USING parameter_name value
```

Note

Once you have created a new database, we recommend that you use the Configure Performance Wizard to obtain recommended values for database manager and database configuration parameters and set them as initial values rather than using default values. The Configure Performance Wizard will suggest suitable values for those parameters based on factors such as workload and server configuration (for example, the number of CPUs). We discuss this wizard in Chapter 2, “Setting up the Control Center” on page 15.

DB2 also provides the registry variables which control the behavior of the database manager. The registry variables can affect both instance level and machine level. We introduce some of the available registry variables which affect the system performance in this book.

To update the registry variables, you can use the following command:

```
db2set variable=value
```

You need to restart the database manager when you change the registry variables.

Before making any changes to performance parameters, be prepared to back out those changes if they do not have the desired effect or have a negative effect on the system. For example, the `db2look` utility with `-f` option extracts the current values of the configuration parameters and the DB2 registry variables. The output of the utility is a script file which you can execute to go back to the setting at the time when the utility was executed. The `db2look` utility also extracts the required DDL statements to reproduce the database objects of a production database on a test database. This utility is very useful when testing against a production database is not possible. See the *DB2 UDB Command Reference,* SC09-2951 for more information.

Note

The `db2look` utility with `-f` option only extracts configuration parameters and registry variables that affect the DB2 query optimizer. Not all registry variables might be extracted.

Apart from changing performance parameters, creating indexes may improve query performance. Defining appropriate indexes may reduce disk I/O and data sorts significantly. You can use the Explain tool to see whether indexes can be used for particular queries.

As we have stated, the major causes of performance problems are actually poor application design or poor database design, rather than the database configuration itself. Before tuning performance parameters, check to make sure that your application or database design is not a cause. Chapter 3, “Data storage management for performance” on page 31 and Chapter 7, “Tuning application performance” on page 229 describe the points you should consider when designing databases and developing applications.

1.6 Architecture and process overview

When working with the performance of the DB2 databases, it is important for you to have a basic understanding of the DB2 architecture and processes. In this section, we provide an overview of the architecture and processes.

Figure 1 shows a diagram of the architecture and processes of DB2 UDB.

Each client application is linked with the DB2 client library and communicates with the DB2 Server using shared memory and semaphores (local clients), or a communication protocol such as TCP/IP and APPC (remote clients).

Note

DB2 Clients cannot use Named Pipe or NetBIOS to communicate with DB2 on UNIX operating systems, including AIX.

On the server side, activity is controlled by engine dispatchable units (EDUs). EDUs are implemented as processes on UNIX, including AIX, and shown as circles or groups of circles in Figure 1.

Note

EDUs are implemented as threads within a single process on Windows 32-bit operating systems and OS/2.

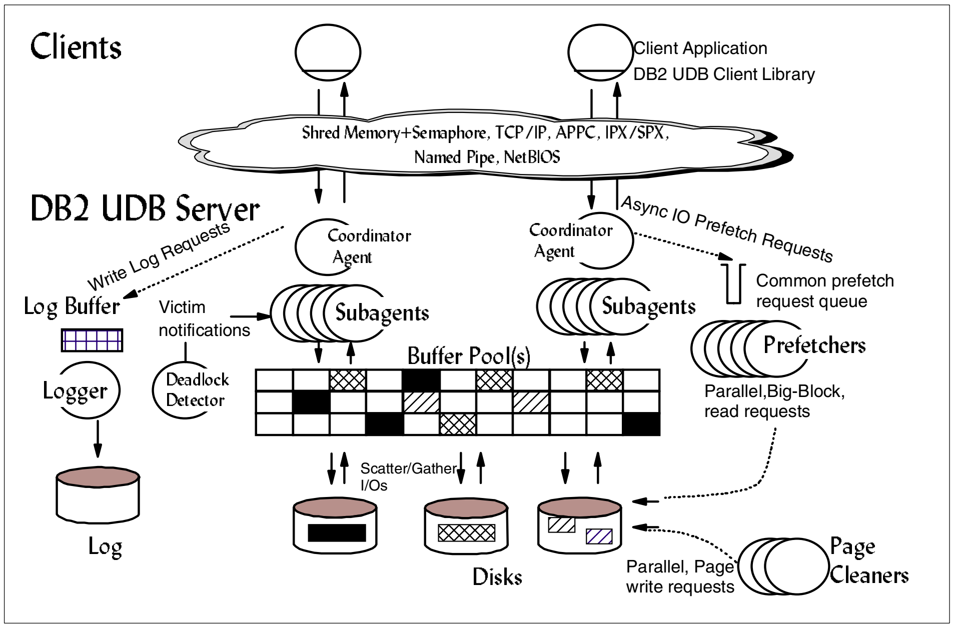


Figure 1. Architecture and process overview

1.6.1 DB2 agents

DB2 agents, including coordinator agents and subagents, are the most common type of DB2 processes which carry out the bulk of the SQL processing on behalf of applications. DB2 assigns a coordinator agent which coordinates the processing for an application and communicates with it.

If you disable intra-partition parallelism, the assigned coordinator agent will pursue all the requests from the application. If you enable intra-partition parallelism, you will see a set of subagents assigned together to the application to work on processing the application requests.

If your database server machine has multiple processors, by letting multiple subagents work together, complex queries requested by the applications can exploit those processors and obtain the result faster. In Figure 1, we assume that intra-partition parallelism is enabled. On UNIX, you can observe coordinator agent processes (`db2agent`) and subagent processes (`db2agntp`) using the `ps` command.

1.6.2 Buffer pools

A buffer pool is an area of storage memory where database pages of user table data, index data, and catalog data are temporarily moved from disk storage. DB2 agents read and modify data pages in the buffer pool. The buffer pool is a key influencer of overall database performance because data can be accessed much faster from memory than from a disk. If more of the data needed by applications were present in the buffer pool, then less time would be needed to access this data, compared to the time taken to find the data on disk storage.

1.6.3 Prefetchers

Prefetchers are present to retrieve data from disk and move it into the buffer pool before applications need the data. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk into the buffer pool if there were no data prefetchers.

Agents of the application send asynchronous read-ahead requests to a common prefetch queue. As prefetchers become available, they implement those requests by using big-block or scatter read input operations to bring the requested pages from disk to the buffer pool. On UNIX, you can see prefetcher processes (`db2pfchr`) by using the `ps` command.

Having multiple disks for storage of the database data means that the data can be striped across the disks. This striping of data enables the prefetchers to use multiple disks at the same time to retrieve data. We discuss disk layout considerations in Chapter 3, “Data storage management for performance” on page 31.

1.6.4 Page cleaners

Page cleaners are present to make room in the buffer pool before prefetchers read pages on disk storage and move into the buffer pool. For example, if you have updated a large amount of data in a table, many data pages in the buffer pool may be updated but not written into disk storage (these pages are called dirty pages). Since prefetchers cannot place fetched data pages onto the dirty pages in the buffer pool, these dirty pages must be flushed to disk storage and become “clean” pages so that prefetchers can place fetched data pages from disk storage.

Page cleaners are independent of the application agents, that look for, and write out, pages from the buffer pool to ensure that there is room in the buffer pool. On UNIX, you can see page cleaner processes (`db2pclnr`) using the `ps` command.

Without the existence of the independent prefetchers and page cleaners, the DB2 agents would have to do all of the reading and writing of data between the buffer pool and disk storage. The configuration of the buffer pool, along with prefetchers and page cleaners, for instance, the size of the buffer pool and the number of prefetchers and page cleaners, control the availability of the data needed by the applications.

Note

When a page cleaner flushes a dirty page to disk storage, the page cleaner removes the dirty flag but leaves the page in the buffer pool. This page will remain in the buffer pool until a prefetcher or a DB2 agent overrides it.

1.6.5 Logs

Changes to data pages in the buffer pool are logged. Agent processes updating a data record in the database update the associated page in the buffer pool and write a log record into a log buffer. The written log records in the log buffer will be flushed into the log files asynchronously by the logger. On UNIX, you can see a logger process (`db2loggr`) for each active database using the `ps` command.

Neither the updated data pages in the buffer pool nor the log records in the log buffer are written to disk immediately to optimize performance. They are written to disk by page cleaners and the logger respectively.

The logger and the buffer pool manager cooperate and ensure that the updated data page is not written to disk storage before its associated log record is written to the log. This behavior ensures that the database manager can obtain enough information from the log to recover and protect a database from being left in an inconsistent state when the database is crashed resulting from an event such as a power failure.

If an uncommitted update on a data page was written to disk, the database manager uses the undo information in the associated log record to undo the update. If a committed update did not make it to disk, the database manager uses the redo information in the associated log record to redo the update. This mechanism is called crash recovery. The database manager performs a crash recovery when you restart the database.

The data in the log buffer is only forced to disk:

- Before the corresponding data pages are being forced to disk. This is called write-ahead logging.
- On a COMMIT; or after the value of the number of COMMITS to group (`mincommit`) database configuration parameter is reached.
- When the log buffer is full. Double buffering is used to prevent I/O waits.

Update data pages

When an agent updates data pages in a buffer pool, these updates must be logged and flushed to disk. The protocol described here minimizes the I/O required by the transaction and also ensures recoverability.

First, the page to be updated is pinned and latched with an exclusive lock. A log record is written to the log buffer describing how to redo and undo the change. As part of this action, a log sequence number (LSN) is obtained and is stored in the page header of the page being updated. The change is then made to the page. Finally, the page is unlatched and unfixed. The page is considered to be dirty because there are changes to the page that have not been written out to disk. The log buffer has also been updated.

Both the data in the log buffer and the dirty data page will need to be forced to disk. For the sake of performance, these I/Os are delayed until a convenient point (for example, during a lull in the system load), or until necessary to ensure recoverability, or to bound recovery time. More specifically, a dirty page is forced to disk when a page cleaner acts on the page as the result of these situations:

- Another agent chooses it as a victim.
- The `CHNGPGS_THRESH` database configuration parameter percentage value is exceeded. Once exceeded, asynchronous page cleaners wake up and write changed pages to disk.
- The `SOFTMAX` database configuration parameter percentage value is exceeded. Once exceeded, asynchronous page cleaners wake up and write changed pages to disk.

We discuss the `CHNGPGS_THRESH` database configuration parameter and the `SOFTMAX` database configuration parameter in Chapter 6, “Tuning configuration parameters” on page 193.

If you set the number of page cleaners to zero and no page cleaner is started, a dirty page is forced to the disk storage by another DB2 agent which chooses it as a victim.

1.6.6 Deadlock detector

A deadlock is a situation in which more than one application is waiting for another application to release a lock on data, and each of the waiting applications is holding data needed by other applications through locking. In such a situation, the applications can wait forever until the other application releases the lock on the held data. The other applications do not voluntarily release locks on data that they need. A process is required to break these deadlock situations.

DB2 uses a background process, called the deadlock detector, to check for deadlocks. When the deadlock detector finds a deadlock situation, one of the deadlocked applications will receive an error code and the current unit of work for that application will be rolled back automatically by DB2. When the rollback is complete, the locks held by this chosen application are released, thereby allowing other applications to continue.

Chapter 2. Setting up the Control Center

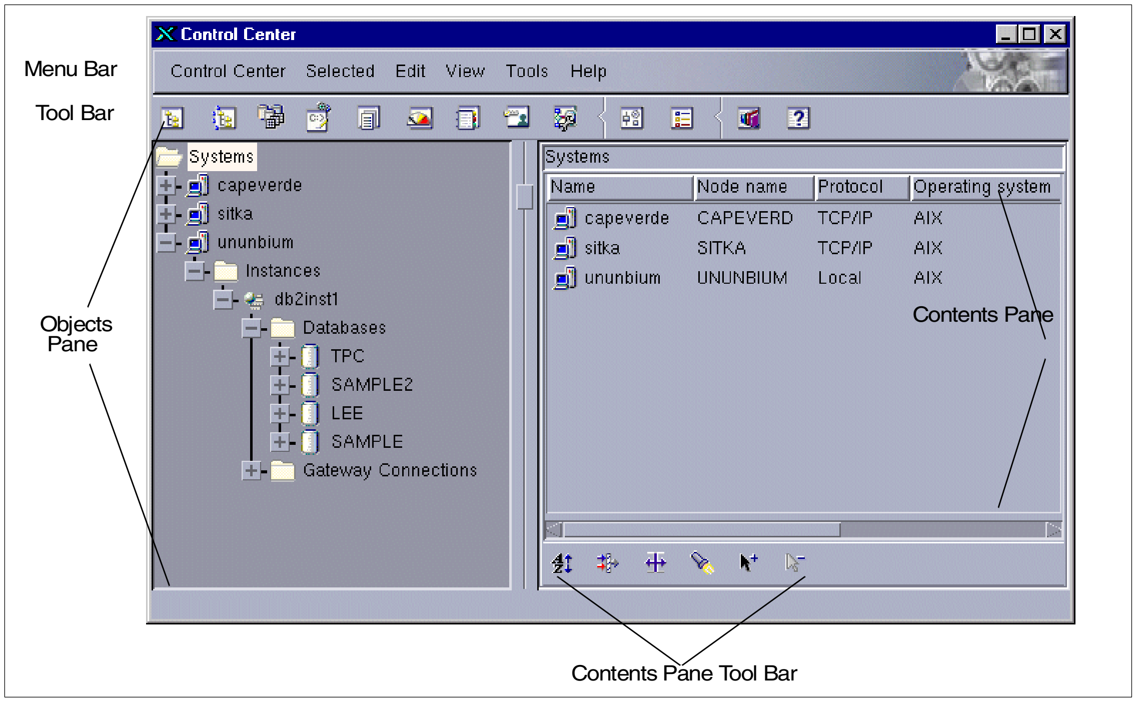
Setting proper values for database manager/database configuration parameters and creating appropriate indexes can achieve a significant performance improvement; however, this task is not easy if you are a new DBA. DB2 UDB provides the Configure Performance Wizard, which helps you to tune performance related configuration parameters by requesting information about the database, its data, and the purpose of the system. For creating indexes, DB2 provides the Index Advisor Wizard, which you can use to determine which indexes to create or drop for a given set of SQL statements.

These wizards or SmartGuides, as they are sometimes called, can be invoked from the Control Center. The Control Center is the central point of administration for the DB2 Universal Database. The Control Center provides the user with the tools necessary to perform typical database administration tasks. It allows easy access to other server administration tools, gives a clear overview of the entire system, enables remote database management, and provides step-by-step assistance for complex tasks.

DB2 provides the install script (`db2setup`), but the Control Center is not installed by default on UNIX platforms. In this chapter we introduce the Control Center and describe how to set up the Control Center.

2.1 Control Center

Figure 2 is an example of the information available from the Control Center. In our example, the Control Center is started at an AIX server, which has one local instance and two remote DB2 server catalog entries.



Systems

Name	Node name	Protocol	Operating system
capeverde	CAPEVERD	TCP/IP	AIX
sitka	SITKA	TCP/IP	AIX
ununbium	UNUNBIUM	Local	AIX

Figure 2. Control Center

The Systems object represents the one local and two remote machines. To display all the DB2 systems that your system has cataloged, expand the object tree by clicking on the plus sign (+) next to Systems. The left portion of the screen lists available DB2 systems. We can see from Figure 2 that the system `ununbium` contains an instance, `db2inst1`. The instance `db2inst1` has four databases.

When Systems is highlighted, details about each system are shown in the Contents Pane. We can see that they are AIX systems.

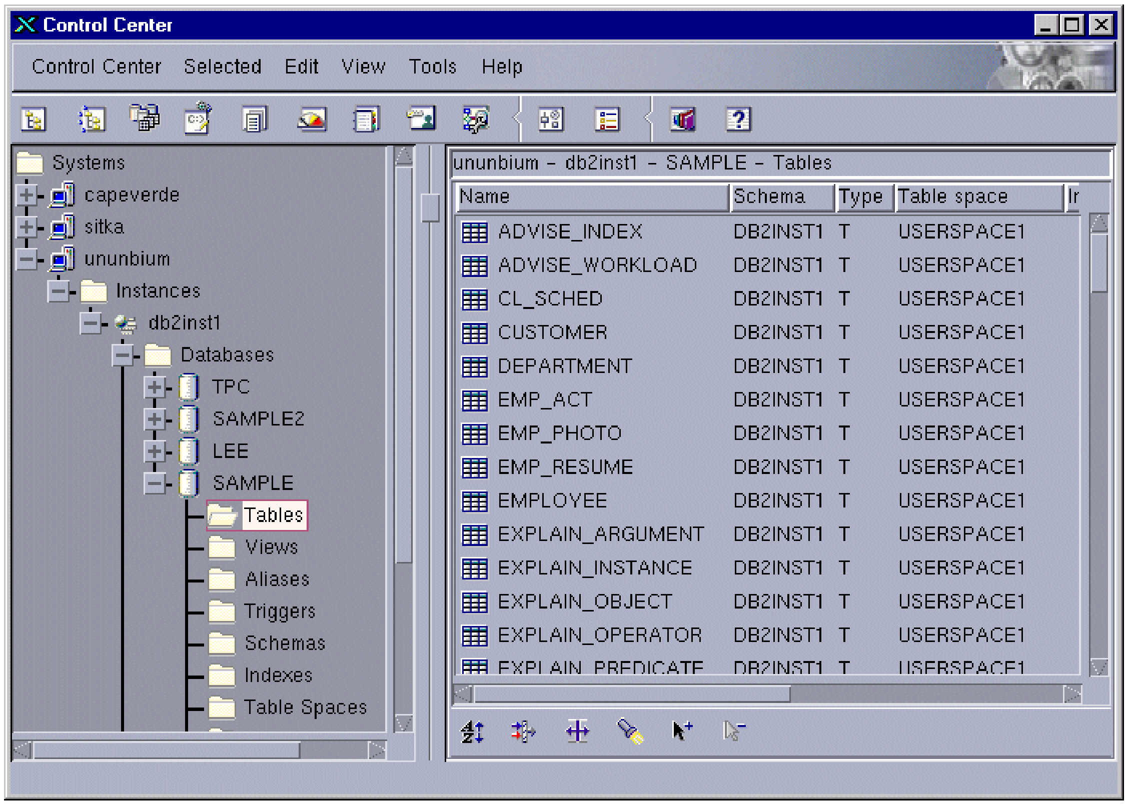
The main components of the Control Center are listed below:

- Menu Bar — This is used to access the Control Center functions and online help.
- Tool Bar — This is used to access the other administration tools.
- Objects Pane — This is shown on the left-hand side of the Control Center window. It contains all the objects that can be managed from the Control Center as well as their relationship to each other.

- Contents Pane — This is found on the right side of the Control Center window and contains the objects that belong or correspond to the object selected on the Objects Pane.
- Contents Pane Toolbar — These icons are used to tailor the view of the objects and information in the Contents pane. These functions can also be selected in the View menu.

Hover Help is also available in the Control Center, providing a short description for each icon on the tool bar as you move the mouse pointer over the icon.

If you want to see each database object in detail, you can expand a database icon and list each database object by clicking on the plus sign (+) next to a database icon. Figure 3 shows the table lists that are defined in the `SAMPLE` database of the `db2inst1` instance.



ununbium - db2inst1 - SAMPLE - Tables

Name	Schema	Type	Table space	Ir
ADVISE_INDEX	DB2INST1	T	USERSPACE1	
ADVISE_WORKLOAD	DB2INST1	T	USERSPACE1	
CL_SCHED	DB2INST1	T	USERSPACE1	
CUSTOMER	DB2INST1	T	USERSPACE1	
DEPARTMENT	DB2INST1	T	USERSPACE1	
EMP_ACT	DB2INST1	T	USERSPACE1	
EMP_PHOTO	DB2INST1	T	USERSPACE1	
EMP_RESUME	DB2INST1	T	USERSPACE1	
EMPLOYEE	DB2INST1	T	USERSPACE1	
EXPLAIN_ARGUMENT	DB2INST1	T	USERSPACE1	
EXPLAIN_INSTANCE	DB2INST1	T	USERSPACE1	
EXPLAIN_OBJECT	DB2INST1	T	USERSPACE1	
EXPLAIN_OPERATOR	DB2INST1	T	USERSPACE1	
EXPLAIN_PREDICATE	DB2INST1	T	USERSPACE1	

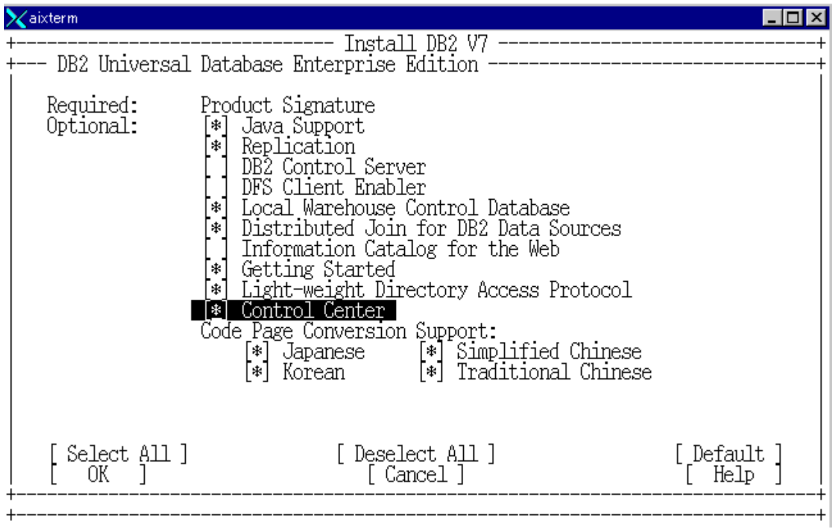
Figure 3. Listing tables using the Control Center

2.2 How to set up the control center

We will now discuss various configurations to manage DB2 running on an AIX machine using the Control Center.

2.2.1 Install the file set

DB2 provides the install utility (`db2setup`) which can perform all of the tasks required to install DB2 but the Control Center is not installed by default. If you want the Control Center to run on the DB2 server, make sure you select the Control Center to be installed. Start the `db2setup` utility, select the option to install the DB2 product, select **Configure**, and check **Control Center** to be installed (Figure 4).



```
+------------------------------- Install DB2 V7 --------------------------------+
+--- DB2 Universal Database Enterprise Edition ---------------------------------+
|
|   Required:     Product Signature
|   Optional:     [*] Java Support
|                 [*] Replication
|                 [ ] DB2 Control Server
|                 [ ] DFS Client Enabler
|                 [*] Local Warehouse Control Database
|                 [*] Distributed Join for DB2 Data Sources
|                 [ ] Information Catalog for the Web
|                 [*] Getting Started
|                 [*] Light-weight Directory Access Protocol
|                 [*] Control Center
|                 Code Page Conversion Support:
|                     [*] Japanese      [*] Simplified Chinese
|                     [*] Korean        [*] Traditional Chinese
|
|
|   [ Select All ]           [ Deselect All ]                [ Default ]
|   [    OK    ]                [ Cancel ]                   [  Help   ]
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
```

Figure 4. The db2setup utility

If you do not use the `db2setup` utility and use the AIX `installp` command or SMIT to install the DB2 products, select the file set `db2_07_01.wcc` to be installed.

The Control Center is written in Java and can be run as a Java application or as a Java applet through a Web server. In both cases, you need a supported Java Virtual Machine (JVM) installed on your machine to run the Control Center. A JVM can be a Java Runtime Environment (JRE) for the Control Center applications, or a Java-enabled browser for the Control Center applets.

On AIX, the `db2setup` utility installs the correct JRE (1.1.8.6 for Version 7.1) for you during DB2 installation if another JRE was not detected on your system. If another JRE was detected on your AIX system during the installation, the JRE that comes with DB2 was not installed. In this case, you must install the correct JRE level before running the Control Center.

For more information about installing the products, please see the manual *DB2 for UNIX Quick Beginnings*,GC09-2970.

2.2.2 Setting up the administration server

The Administration Server (DAS) instance responds to requests from the Control Center and other DB2 Administration Tools. The DAS instance must be running on every DB2 server that you want to administer using the Control Center. You can create the DAS instance using the `db2setup` utility during the product installation. The `dasicrt` command under `/usr/lpp/db2_07_01/instance` directly can also be used to create the DAS instance.

The DAS instance is started automatically. You can start and stop it using the `db2admin start` and `db2admin stop` command.

2.2.3 Run the Control Center locally or remotely

There are two methods to use the Control Center to manage DB2. One is installing and starting the Control Center on the server and managing local DB2 instances and databases. The other one is installing and starting the Control Center on a DB2 client on which the databases on the server are cataloged, and managing them as remote DB2 instances and databases. For example, you can manage the instances and databases created on an AIX server machine using the Control Center running on a Windows NT client.

If you want to use the Control Center on the client machines, be aware that the DB2 Run-Time Client does not provide the GUI tools. You should install the DB2 Administration Client or DB2 Software Developer's kit. In this case, the process of the Control Center is running on the client machine, and the instances and databases on the server machine are managed remotely.

2.2.4 Java application or Java applet

Java applications run just like other applications on your machine, provided that you have the correct JRE installed. As already stated, on AIX, the `db2setup` utility installs the correct JRE (1.1.8.6 for Version 7.1) for you during DB2 installation if another JRE was not detected on your system.

If you intend to use the Control Center on Windows 32-bit operating systems, the correct JRE level is installed or upgraded for you, and therefore you do not need to be concerned about the JRE level.

If you intend to use the Control Center on other operating systems than Windows and AIX, you must install the correct JRE level before running the Control Center. See the manual *Installation and Configuration Supplement*, GC09-2957 for information about the correct JRE level you need to install. Also, check the latest service information on the Control Center at the following URL:

`http://www.ibm.com/software/data/db2/udb/db2cc/`

Java applets are programs that run within Java-enabled browsers. The Control Center applet code can reside on a remote machine and is served to the client's browser through a Web server. Since there are no supported browsers for AIX and other UNIX operating systems, this is not an option you can choose if you want to use the Control Center from UNIX machines. However, on Windows 32-bit or OS/2 operating systems, you can use the Control Center as a Java applet using a supported Java-enabled browser. See the manual *Installation and Configuration Supplement,* GC09-2957 for information about the supported browser.

2.2.5 Machine configuration scenario

As already discussed, you can run the Control Center to manage local databases or remote databases, and you can also run it as a Java application or as a Java applet. Therefore, you can select one of the four configuration scenarios shown in Table 1, to use the Control Center. This table shows different ways of installing the required components.

Table 1. The control center machine configuration scenarios

Scenario	Machine A	Machine B	Machine C
1.Stand-alone Java Application	DB2 Server Control Center (Java application) JDBC Applet Server JRE		
2.Two-tier Java Application	DB2 Admin. Client Control Center (Java application) JDBC Applet Server JRE		DB2 Server

Scenario	Machine A	Machine B	Machine C
3.Two-tier Java Applet	Supported Browser (Windows, OS/2 only)	DB2 Server Control Center (Java applet) JDBC Applet Server Web server	
4.Three-tier Java Applet	Supported Browser (Windows, OS/2 only)	DB2 Admin. Client Control Center (Java applet) JDBC Applet Server Web server	DB2 Server

Scenario 1 and Scenario 2 are cases that the Control Center runs as a Java application. In Scenario 1, the Control Center runs on the same machine as the DB2 server runs. In Scenario 2, the Control Center runs on a different machine as the DB2 server runs and accesses it through the DB2 Administration Client. Machine A can be any supported platform for the DB2 Administration Client.

Scenario 3 and Scenario 4 are cases that the Control Center runs as a Java applet. In Scenario 3, the Control Center applet code resides on the same machine as the DB2 server runs, and it is served to the client's browser through a Web server. In Scenario 4, the Control Center applet code is served to the client's browser through a Web server as well; however, it resides on a different machine in which the DB2 server runs and access it through the DB2 Administration Client. In Scenarios 3 and 4, only a Windows or OS/2 machine can be Machine A.

2.2.6 Control Center services setup

In all scenarios, the DB2 JDBC Applet Server must be started with a user account that has SYSADM authority on the machine where the DB2 JDBC Applet Server resides. You can set your DB2 JDBC Applet Server to start automatically at startup time.

On AIX or other UNIX operating systems containing the DB2 JDBC Applet Server, enter the `db2jstrt 6790` command.

On the Windows 32-bit or OS/2 operating system containing the DB2 JDBC Applet Server, enter the `db2jstrt 6790` command to start the DB2 JDBC Applet Server. On Windows NT, you can also start the DB2 JDBC Applet Server by clicking on **Start**->**Control Panel**->**Services**. Select the **DB2 JDBC Applet Server - Control Center** service and click on the **Start** button.

Note

The number 6790 specified is the port number that the DB2 JDBC Applet Server is using; however, this is just an example (and default value). Any number between 1024 and 65535 that is not already in use can be used.

2.2.7 Start the Control Center as a Java application

To start the Control Center as a Java application, execute the `db2cc` command from the command prompt. If you have started the DB2 JDBC Applet Server with a port number other than the default (6790), enter the new port number after the `db2cc` command, such as `db2cc 6789`.

2.2.8 Start the Control Center as a Java applet

To run the Control Center as a Java applet, you must have a Web server set up on the machine that contains the Control Center applet code and the DB2 JDBC Applet Server. The Web server must allow access to the `sqllib` directory.

If you choose to use a virtual directory, substitute this directory for the home directory. For example, if you map `sqllib` to a virtual directory called `temp` on a server named `yourserver`, then a client would use the URL:

```
http://yourserver/temp
```

For more information on installing and configuring the Control Center as a Java applet, see the manual *Installation and Configuration Supplement*, GC09-2957.

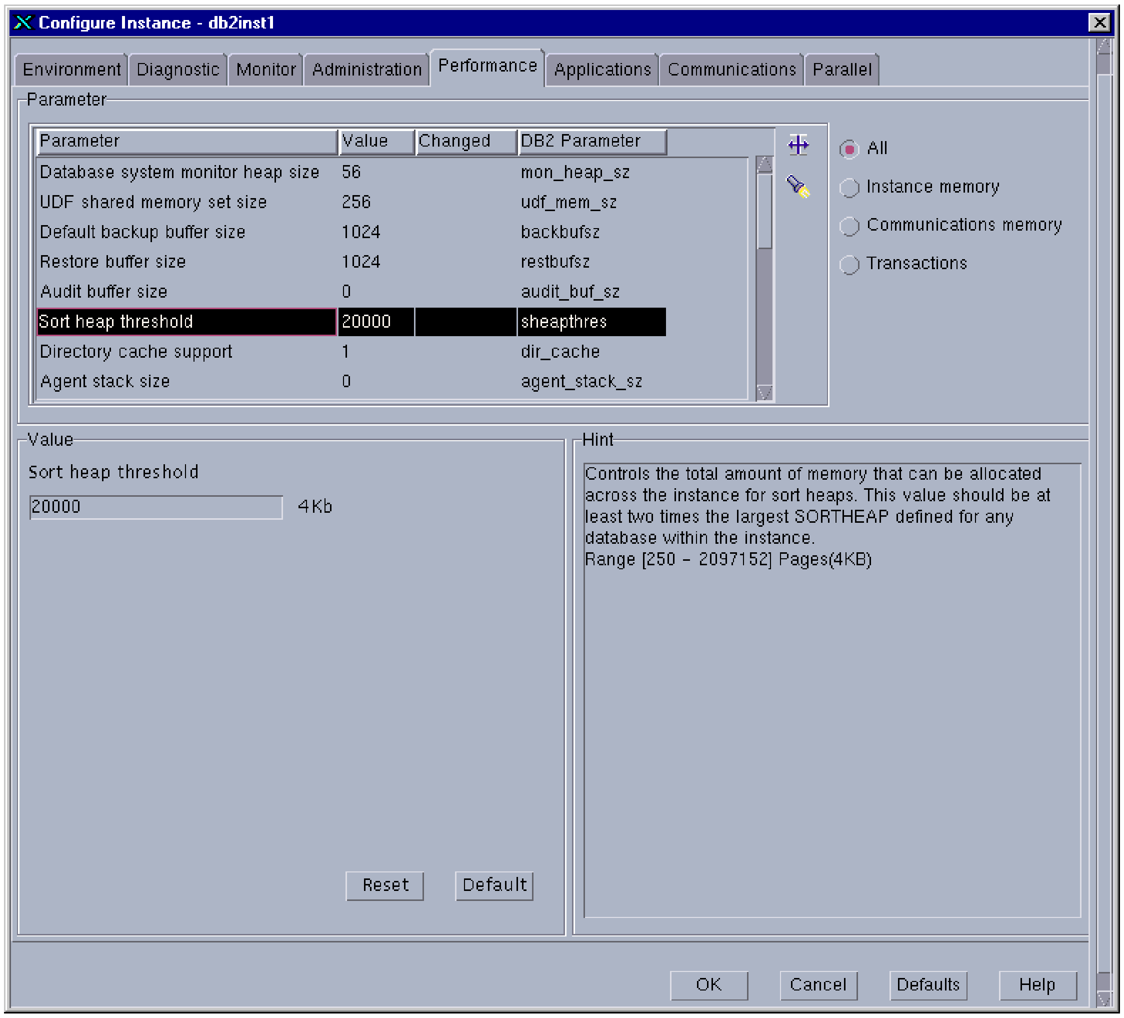
2.3 Performance tuning using Control Center

You can tune the database manager and database configuration parameters from the Control Center. You can specify the values for each parameter through the GUI panel, or you can use the Configure Performance Wizard to obtain recommended values and apply them. Here we introduce both methods.

2.3.1 Setting up configuration parameters

The Control Center provides you the GUI panel to change the database manager and database configuration parameters. To bring up the panel to update the database manager configuration parameters, right-click on an

instance icon in the Control Center and select **Configure**. You will see the Database Manager Configuration Notebook that displays current values of the database manager configuration parameters grouped by function. Each notebook page has the Parameters controls, a Value field to change the value of the selected configuration parameter, and a Hint box that gives a brief description of what the configuration parameter is used for. Figure 5 shows the Performance page that contains the database manager configuration parameters that affect the performance of the DB2 instance.



Parameter	Value	Changed	DB2 Parameter
Database system monitor heap size	56		mon_heap_sz
UDF shared memory set size	256		udf_mem_sz
Default backup buffer size	1024		backbufsz
Restore buffer size	1024		restbufsz
Audit buffer size	0		audit_buf_sz
Sort heap threshold	20000		sheapthres
Directory cache support	1		dir_cache
Agent stack size	0		agent_stack_sz

Value

Sort heap threshold

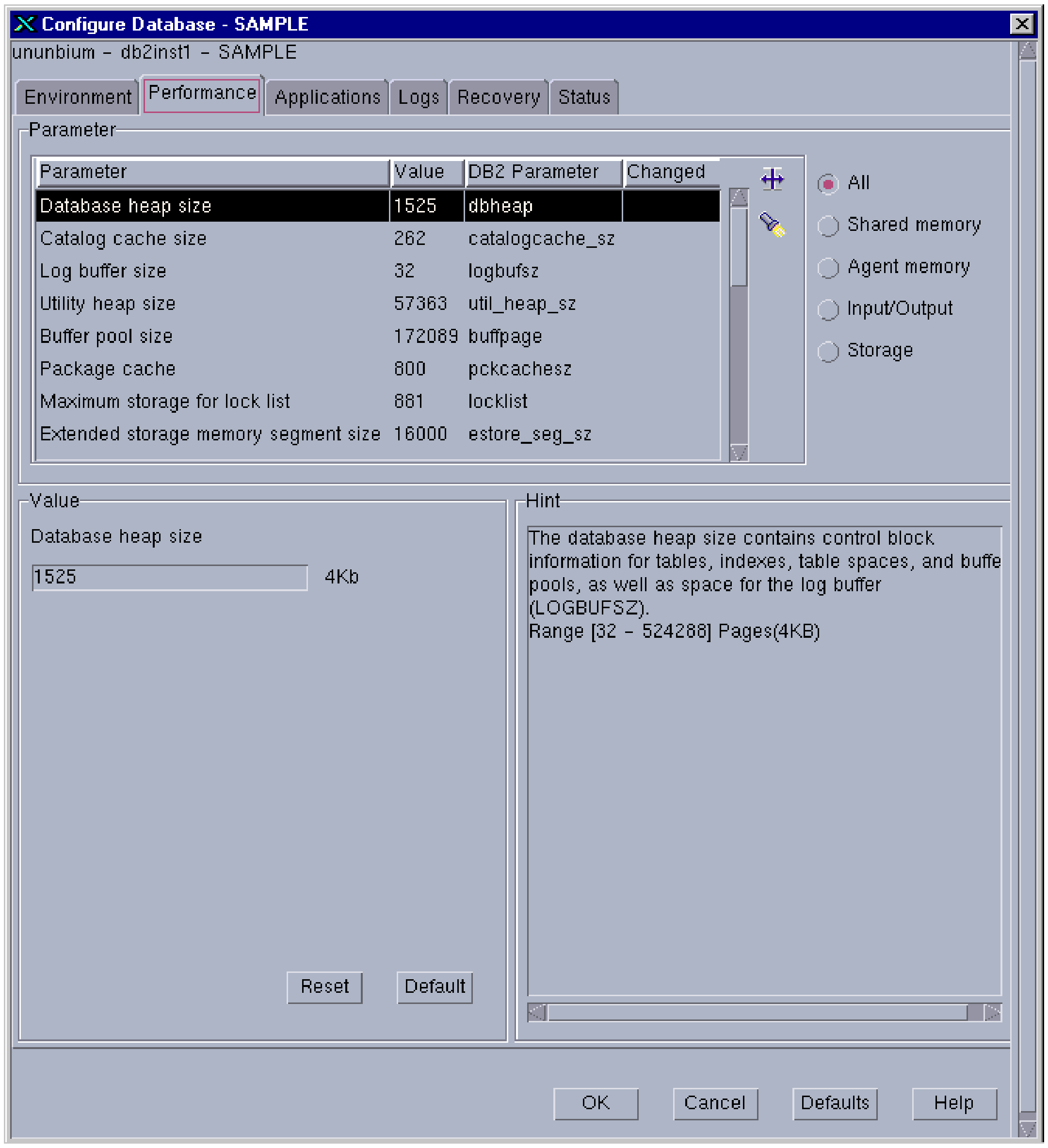
20000 4Kb

Hint

Controls the total amount of memory that can be allocated across the instance for sort heaps. This value should be at least two times the largest SORTHEAP defined for any database within the instance.
Range [250 - 2097152] Pages(4KB)

Figure 5. The database manager configuration notebook

To bring up the panel to update the database configuration parameters, right-click on a **database** icon in the Control Center and select **Configure**. You will see the Database Configuration Notebook that displays current values of the database configuration parameters grouped by function. Each notebook page has the Parameters controls, a Value field to change the value of the selected configuration parameter, and a Hint box that gives a brief description of what the configuration parameter is used for. Figure 6 shows the Performance page that contains the database configuration parameters that affect the performance of the SAMPLE database.



Parameter	Value	DB2 Parameter	Changed
Database heap size	1525	dbheap	
Catalog cache size	262	catalogcache_sz	
Log buffer size	32	logbufsz	
Utility heap size	57363	util_heap_sz	
Buffer pool size	172089	buffpage	
Package cache	800	pckcachesz	
Maximum storage for lock list	881	locklist	
Extended storage memory segment size	16000	estore_seg_sz	

Database heap size

1525 4Kb

The database heap size contains control block information for tables, indexes, table spaces, and buffe pools, as well as space for the log buffer (LOGBUFSZ).
Range [32 – 524288] Pages(4KB)

Figure 6. The database configuration notebook

2.3.2 Using the Configure Performance Wizard

If you are a new DBA or someone who only administers a database occasionally, the Configure Performance Wizard is a very useful tool to tune the configuration parameters. Even if you are an experienced DBA, we recommend that you use the Configure Performance Wizard when you start building a new database and obtain its recommendation. You should start the recommended values instead of the default values of the database manager and database configuration parameters.

To use the Configure Performance Wizard, perform the following steps:

1. Expand the Object Tree to display **Databases** folder.
2. Right-click database to be changed, select **Configure Performance Using Wizard** from pop-up menu or from the **Selected** option on the menu bar. This is shown in Figure 7.

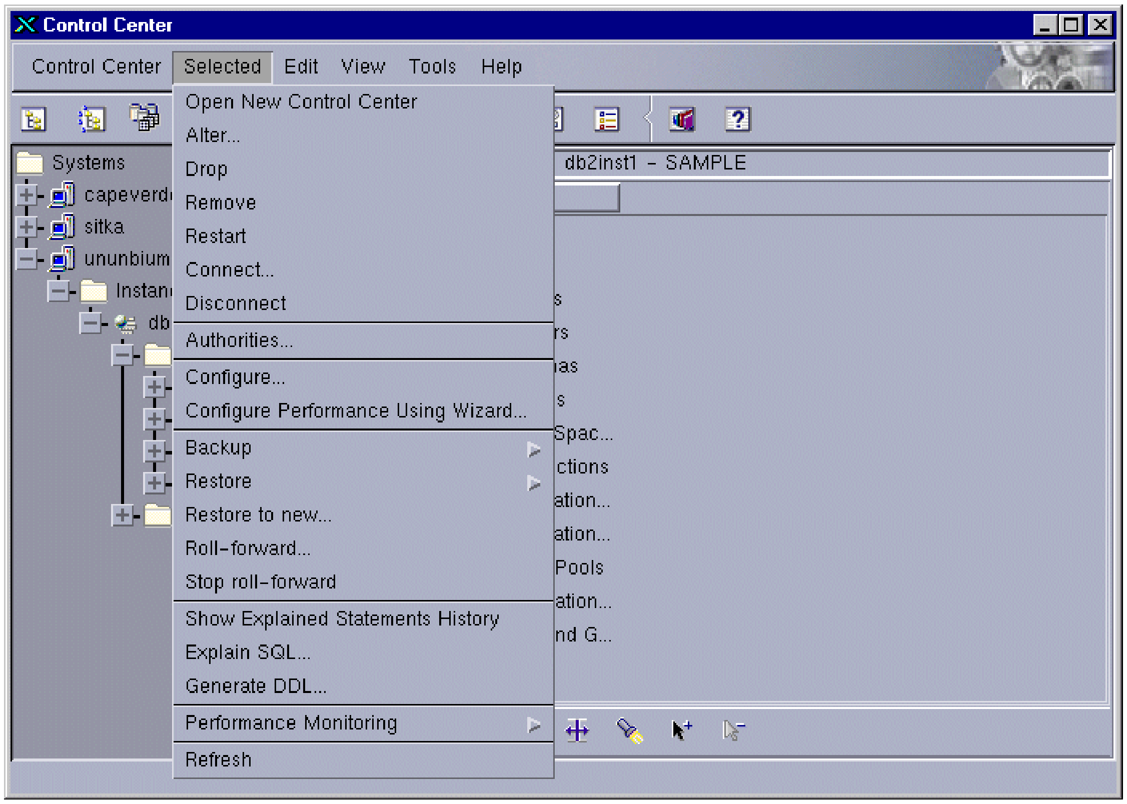
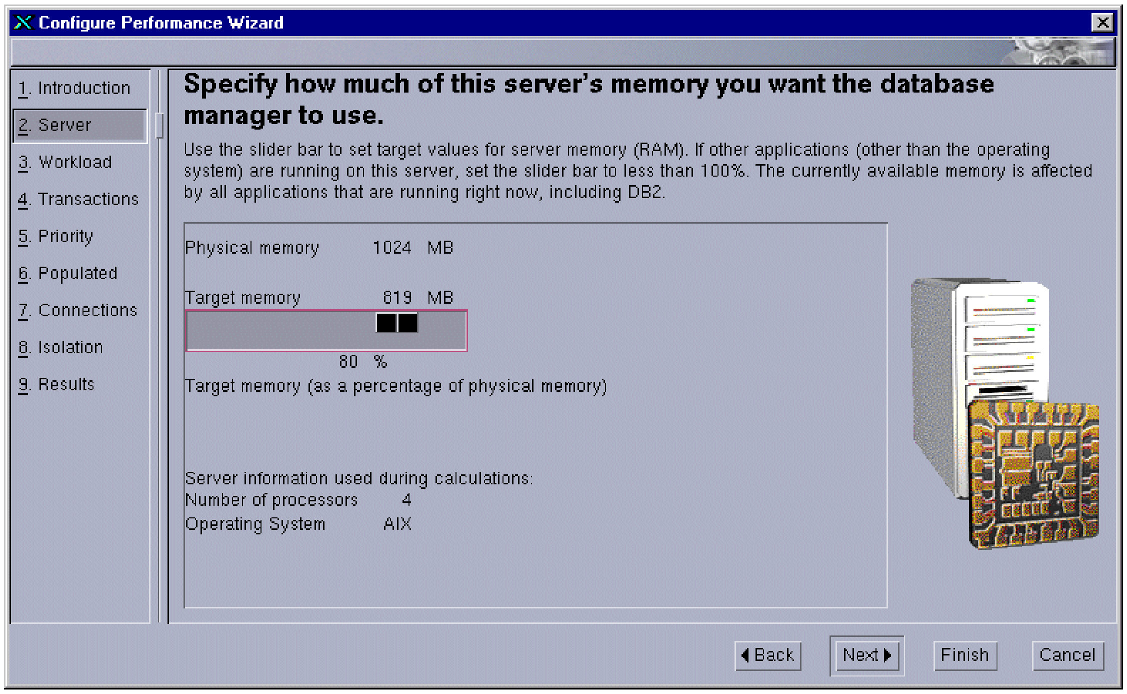


Figure 7. Configure Performance Wizard — starting

3. Work through each page, changing information where necessary. The Configure Performance Wizard will calculate and provide the appropriate value of each database manager and database configuration parameter based on information you supply. The configuring operation using the Configure Performance Wizard consists of 7 steps. You should enter or select the value in each step. Figure 8 shows the first step, in which you need to enter how much of this server's memory you want the database manager to use.



Specify how much of this server's memory you want the database manager to use.

Use the slider bar to set target values for server memory (RAM). If other applications (other than the operating system) are running on this server, set the slider bar to less than 100%. The currently available memory is affected by all applications that are running right now, including DB2.

Physical memory 1024 MB

Target memory 819 MB

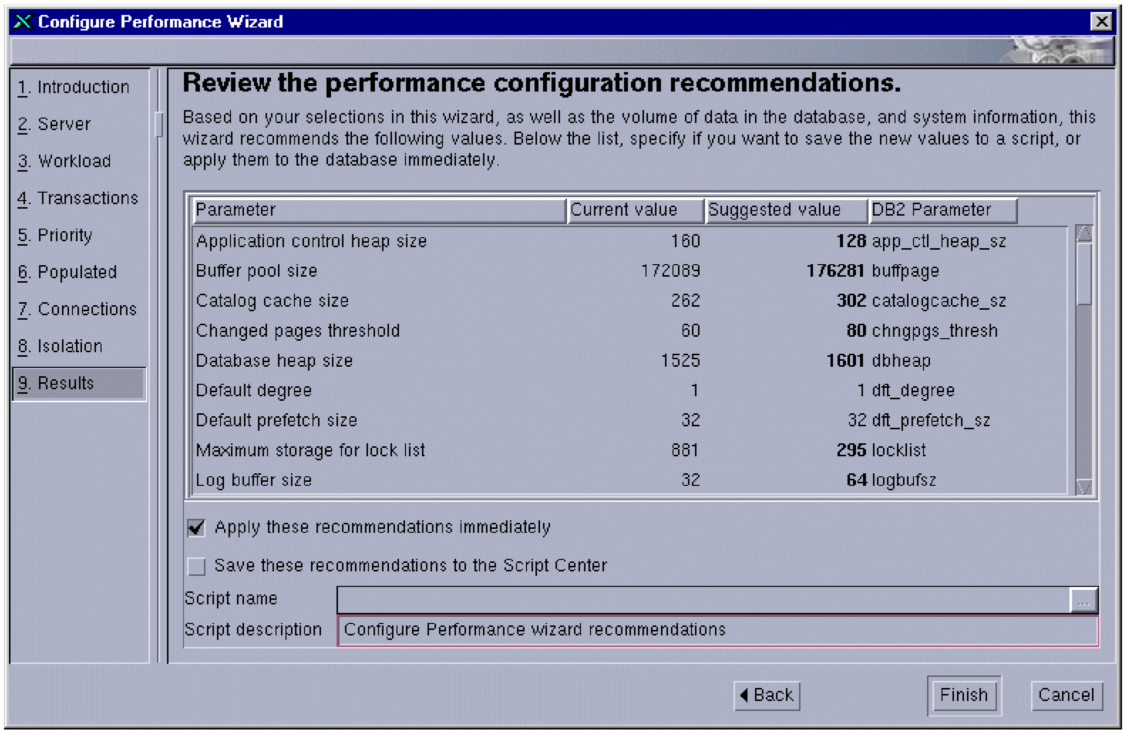
80 %

Target memory (as a percentage of physical memory)

Server information used during calculations:
Number of processors 4
Operating System AIX

Figure 8. Configure Performance Wizard — input values

4. The Configure Performance Wizard calculates the appropriate value of each database manager and database configuration parameter when you finish entering the value of all steps. An example of the Results page can be seen in Figure 9. It shows the current value and suggested value of each configuration parameter. You can apply these recommendations immediately or save to Script Center to apply them later.



Review the performance configuration recommendations.

Based on your selections in this wizard, as well as the volume of data in the database, and system information, this wizard recommends the following values. Below the list, specify if you want to save the new values to a script, or apply them to the database immediately.

Parameter	Current value	Suggested value	DB2 Parameter
Application control heap size	160	**128**	app_ctl_heap_sz
Buffer pool size	172089	**176281**	buffpage
Catalog cache size	262	**302**	catalogcache_sz
Changed pages threshold	60	**80**	chngpgs_thresh
Database heap size	1525	**1601**	dbheap
Default degree	1	1	dft_degree
Default prefetch size	32	32	dft_prefetch_sz
Maximum storage for lock list	881	**295**	locklist
Log buffer size	32	**64**	logbufsz

[✓] Apply these recommendations immediately

[ ] Save these recommendations to the Script Center

Script name	
Script description	Configure Performance wizard recommendations

Figure 9. Configure Performance Wizard — results

2.3.3 Is this all I need to do for performance tuning?

The best way to think of the recommendations made by the Configure Performance Wizard is as a starting point when you create a new database, or when an existing database has been changed significantly and you need to find better values for configuration parameters. Use the suggested values as a base on which to make further adjustments as you try to optimize the performance of your database. We hope that this book will aid you in this regard.

2.3.4 Index Advisor Wizard

Creating appropriate indexes is crucial for optimal database performance, but this is a difficult task for a new DBA. To help with this task, DB2 provides the Index Advisor Wizard, which will determine the best set of indexes for a given workload.

We discuss the Index Advisor Wizard in Chapter 3, “Data storage management for performance” on page 31.

2.4 DB2 UDB wizards

Besides the Configure Performance Wizard and the Index Advisor Wizard, DB2 provides the following wizards:

- **Backup Database** — This wizard asks you basic questions about the data in the database, the availability of the database, and recoverability requirements. It then suggests a backup plan, creates the job script, and schedules it.
- **Create Database** — This wizard allows you to create a database, assign storage, and select basic performance options.
- **Create Table space** — This wizard lets you create a new table space and set basic storage performance options.
- **Create Table** — This wizard helps you to design columns using predefined column templates, create a primary key for the table, and assign the table to one or more table spaces.
- **Restore Database** — This wizard walks you through the process of recovering a database.
- **Configure Multi-Site Update** — This wizard lets you configure databases to enable applications to update multiple sites simultaneously when it is important that the data at all the sites must be consistent.

Chapter 3. Data storage management for performance

When working with relational databases, the main objective is to be able to store and retrieve data quickly and efficiently. One important consideration when trying to design a new database or analyze a performance problem on an existing database is the physical layout of the database itself. The placement of data can directly affect the performance of the operation you are trying to perform, be it a query, load, import, export or backup. The overall objective is to spread the database effectively across as many disks as possible to try and minimize I/O wait. It is essential that the database administrator understands the different advantages and disadvantages of choosing a particular data placement strategy in order to achieve this goal.

In this chapter we will discuss the following concepts:

- Disk layout
- Mirroring, striping, and RAID devices
- Table space design
- Buffer pool layout
- Database logs

We will explain how each of these concepts effects database performance, and provide helpful information that a database administrator can use to physically design a new database or improve the physical design of an existing database.

We will not explain basic concepts and tasks, such as creating a database, a table space, or a buffer pool. Rather, we will provide information that can be used to help understand how creating these components in certain ways can affect performance.

For information on the commands to perform these tasks, please refer to the *DB2 UDB Administration Guide: Implementation*, SC09-2944 or the *DB2 UDB SQL Reference*, SC09-2974.

3.1 Disk layout

Deciding how to manage disk storage when working with any relational database is one of the most important decisions a database administrator has to make.

How the data is physically laid out across the disks affects performance, as we explain in the following sections.

3.1.1 Data placement

When choosing a physical disk layout, a number of factors come into play:

Workload considerations
The type of workload your database is tasked to perform needs to be taken into account when considering how the physical database layout may affect the performance of the database. The primary function of each workload type should give you some indiction of how to lay out the database.

For example, if the database is part of a Web application, then typically its behavior will be On-line Transaction Processing (OLTP). There will be a great deal of concurrent activity, such as large numbers of sessions active, simple SQL statements to process, and single row updates. Also, there will be a requirement to perform these tasks in seconds.

If the database is part of a Decision Support System (DSS), there will be small numbers of large and complex query statements, fewer transactions updating records as compared to transactions reading records, and many sequential I/O operations.

This information gives the database administrator some ideas which can be used to aid in the physical design of a database which will be well-suited to that type of workload.

Available disk space
You need to consider whether you have enough disk space to support the raw data. Also, remember that additional space is required for indexes, temporary space, secondary log files, and paging space. The total amount of required disk space for all of these is, in most cases, three to four times the raw data; this needs to be calculated before you even consider allowances for concepts such as mirroring. If you then wish to introduce mirroring, eight times the raw data would be a good estimate. RAID 5 requirements are somewhere between these two situations.

Number of disks required
It is important also to take into account the number of disks required to achieve a balanced system. For example, a system made up of many small disks can be exposed to potential problems, such as those caused by system bus contention; such a system may also produce increased system administration overhead. The opposite of this, having too few large disks, can also cause I/O bottlenecks. In an average system, a good balance would be a minimum of six to ten disks per CPU for optimal performance.

In summary, the basic goal is to design a system so that no one disk or set of disks becomes a performance bottleneck.

Running out of disk space

Reaching your disk storage limit can affect overall performance and, in the worst case, the operation of the database (if you run out of log space).

3.1.2 Log placement

Factors affecting logging performance such as log file size, number of logs and type of containers used for logs will be discussed in 3.5, “Database logs” on page 78. However before any log files ever exist the database administrator needs to decide on two main factors:

On which disks are the logs to be placed?

Due to the importance of the log files, it is recommended that these should ALWAYS reside on their own physical disk(s), separate from the rest of the database objects. The disks ideally should be dedicated to DB2 logging to avoid the possibility of any other processes accessing or writing to these disks.

Another important factor is the location of the logical volume on the disk itself. In most cases, the general advice would be to place the logical volume as close to the center of the disk as possible. However, with the log filesystem or raw logical volumes, this is NOT the case. The typical activity of the DB2 logger process, which writes to the log files, will be a large number of sequential reads and writes. For this reason, when creating the logical volume, ideal placement is on the outer edge of the disk where there are more data blocks per track.

Availability

Irrespective of the type of logging you choose, whether it be Circular Logging or Archival Logging, the availability of the physical disks is crucial to the database. For this reason, it is strongly recommended that you protect the log against single disk failures by using RAID 1 (mirroring the log devices), or by storing log files on a RAID 5 array.

You might wonder why a RAID 5 array, which incurs the write penalty, is suitable for log files. We will discuss RAID disks and write penalty in 3.2, “Mirroring, striping, and using RAID devices” on page 35.

3.1.3 Data availability and performance

In these days of e-business, with DB2 databases commonly being used to support Web applications, 24x7 availability is an important consideration. It is worth remembering that in most cases, increasing availability is a trade-off against a potential reduction in performance. The most common solutions implemented today are RAID 5, discussed in 3.2, “Mirroring, striping, and using RAID devices” on page 35, and data mirroring.

It is also worth mentioning that in an attempt to improve I/O performance to frequently accessed tables, many customers choose to implement data striping across logical volumes. Mirroring and striping will also be covered in 3.2, “Mirroring, striping, and using RAID devices” on page 35. Although obviously, both mirroring and striping can be done outside of the RAID array by the operating system’s logical volume manager, the concepts are the same.

3.1.4 General performance recommendations

The following are general performance considerations to bear in mind when creating logical volumes on AIX:

- Try to spread heavily-accessed logical volumes across as many physical volumes as possible to enable parallel access.
- Try to create your logical volumes of sufficient size for your data; this will avoid fragmentation if the logical volumes have to be increased later.
- Sequential write-intensive logical volumes are best placed on the outer edge of the physical volumes. Logical volume placement can be controlled during its creation.
- Sequential read-intensive logical volumes are best placed in the center of the physical volumes.
- Set inter-policy to maximum to spread each logical volume across as many physical volumes as possible.
- Set write verify to OFF; see 3.2.3, “AIX logical volume parameters” on page 40.

Note

Some further considerations for mirroring and striping are also discussed in 3.3, “Table spaces: performance considerations” on page 52

3.2 Mirroring, striping, and using RAID devices

RAID levels can be categorized into different degrees of performance and reliability. Certain RAID levels (0 and 1) are known as striping and mirroring respectively; we mention this now, as data striping and mirroring are not confined to RAID arrays. These functions can be performed outside of RAID using the operating system's logical volume manager. However, the overview of each is still applicable, regardless of whether or not the logical volume resides on RAID disks or is created outside of the RAID array using the AIX Logical Volume Manager (LVM).

3.2.1 Summary of the most popular RAID levels

Each of the RAID levels supported by disk arrays uses a different method of writing data, and hence provides different benefits.

3.2.1.1 RAID 0

Also known as striping, RAID 0 does not bring any additional data redundancy to a disk subsystem. Primary goal of RAID 0 is to improve I/O performance. When data is written, using a striped logical volume, the data is divided into small pieces known as chunks or stripe units. These units are then written to the disks in parallel, in a round-robin fashion. During a read operation the same units are read back in parallel then written directly to the caller's memory area where they are re-assembled into the original data.

Main problem with RAID 0 is obviously that any disk failure in the array will render the data on the remaining disks unusable. For that reason only consider RAID 0 is you require good performance but are not concerned about availability.

3.2.1.2 RAID 1

Also known as mirroring, RAID 1 is simply keeping a copy of the data in a different location to protect against data loss in the event of a disk failure. So from the availability perspective, the copies should be kept on a separate physical volume which ideally should be attached to a separate I/O adapter. To take this one stage further, another possibility would be to have the second disk in a separate disk cabinet to protect against events such as a power loss to one of the disk cabinets.

In the event of a disk failure, read and write operations will be directed to the mirrored copy of the data. In performance terms, mirroring has some advantages over striping. If a read request is received and the primary data copy is busy, then the AIX LVM (Logical Volume Manager) will read the data from the mirrored copy. Write performance is affected by the write scheduling policy configured using the AIX LVM options, which are `PARALLEL` or `SEQUENTIAL`.

Although mirroring can improve the read performance, the penalty is write performance. For optimal performance, we recommend the use of the Parallel scheduling policy. See 3.2.3, “AIX logical volume parameters” on page 40 for more details on scheduling policy.

- For more detailed information on the AIX Logical Volume Manager, refer to your AIX documentation.

Note

In AIX, when mirroring data, ONLY the logical volumes are mirrored, not the physical disk. Mirroring is not a tool to provide a disk copy

3.2.1.3 RAID 5

In a RAID 5 array, data and parity are spread across all disks. For example, in a 5+P RAID 5 array, six disks are used for both parity and data. In this example, five-sixths of the available space is used for data and one-sixth is used for parity.

Because of the parity data used by RAID 5, each write to the array will result in four I/O operations; this is known as the RAID 5 write penalty:

1. Read old data
2. Read old parity
3. Write new data
4. Write new parity

Fast Write Cache (FWC), which exists on some RAID adapters, can reduce the effects of this write penalty. RAID 5 is commonly chosen because it provides an extremely good price/performance ratio combined with good availability. By default, FWC will be OFF; see 3.2.4, “SSA RAID array parameters” on page 42 for more information on checking and setting FWC.

Using the Enterprise Storage Server (ESS), which has a large cache, can significantly decrease the effects of the write penalty. See 3.2.9, “Using Enterprise Storage Server” on page 50 for more information on ESS.

3.2.1.4 RAID 10

- Sometimes called RAID 0+1, this RAID level provides better data availability at the cost of extra disks. Consider a striped logical volume, where failure of one disk renders the entire logical volume unusable. RAID

10 provides the ability to mirror the striped logical volumes to prevent this. When data is written to the disks, the first data stripe is data and the subsequent stripe copies (maximum of three copies, including first copy) are the mirrors and are written to different physical volumes.

Note

AIX 4.3.3 introduces the ability to create logical volumes outside of RAID which are mirrored and striped, known as RAID 0+1 or RAID 10.

3.2.2 Performance considerations

If you are going to be using striping, mirroring or RAID, remember that each of these concepts has its own set of advantages and disadvantages to be considered, as well as tips which can be used to help obtain optimal performance. In this section, we will provide an overview of these points.

3.2.2.1 Striping

If you are using striping, the following are some general performance considerations:

- Set `max_coalesce` equal to or a multiple of the stripe unit size, but not smaller. See 3.2.4, “SSA RAID array parameters” on page 42.
- Set `minpgahead` to 2 and `maxpgahead` to 16 * number of disk drives, using the AIX `vmtune` command. Refer to the AIX documentation for more details on these parameters.
- Modify `maxfree` to accommodate the change in `maxpgahead` so that the difference between `minfree` and `maxfree` is `maxpgahead`.
- If the striped logical volumes are raw logical volumes, consider increasing the `lvm_bufcnt` parameter of the AIX `vmtune` command; see 3.2.3, “AIX logical volume parameters” on page 40 for more details on this.
- If you create a logical volume for a DB2 container which will be intensively used and updated, try NOT to put it on a disk with your striped logical volumes.

Note

The `vmtune` parameters `maxpgahead` and `maxpgahead` only affect I/O which goes through the AIX LVM; raw device I/O is not affected.

To use `vmtune`, install the AIX fileset `bos.adt.samples`, and the executable will be placed in the `/usr/samples/kernel/` directory

3.2.2.2 Mirroring

The following are performance considerations for mirroring:

- Be aware that when you use mirroring, every write performs multiple writes, this will affect performance. When mirroring, set the scheduling policy to `parallel` and allocation policy to `strict`, as discussed in 3.2.3, “AIX logical volume parameters” on page 40.
- Mirrored logical volumes which are frequently accessed should be placed on the outer edge of the physical volume, because the mirror write consistency cache resides there and this is updated frequently. Please refer to the 3.2.3.3, “Mirror write consistency” on page 41 for more details.

3.2.2.3 RAID 5 versus LVM mirroring

SSA RAID 5 or LVM mirroring are the most widely implemented data availability solutions used in most DB2 systems. The main reason for this is that both solutions provide a robust and reliable data protection solution for a reasonable price/performance ratio. To decide between RAID 5 and LVM mirroring, consider these factors:

RAID 5:
Read performance is affected by number of disks within the array. With a greater number of disks, the I/O throughput increases, because more requests can be processed in parallel (up to the RAID adapter limit).

If your database application is very write-intensive, then the write penalty associated with RAID 5 will become a factor. RAID adapters possessing the fast write cache feature can help minimize that penalty. If your application generates mostly read operations, then a RAID 5 array will give performance similar to a mirrored system; to do this, the array will need to use enough disks to provide adequate parallel reads.

Mirroring:
The main factor with a mirrored system is the number of individual disks required to store the data. For example, data stored on a RAID array consisting of 11 disks (10 parallel + 1 parity) would require 20 disks for a mirrored solution.

3.2.2.4 Log files on a RAID 5 array

Although a RAID 5 array may incur the write penalty, when you have FWC or use the Enterprise Storage Server (ESS), we recommend that you use a RAID 5 array to store log files rather than using mirrored disks. The benefit you can obtain from a using a RAID array is enough to justify the overhead due to writing the parity information, for the following reasons:

- The typical activity on the log files is a large number of sequential reads and writes; therefore, all of the disks in the RAID array can work in parallel and provide a higher bandwidth than mirrored disks can provide.
- With RAID-5, the amount of data to be stored in the disk storage is smaller than with mirrored disks, which means less utilization of the disk and memory buses.
- When a sequential write occurs, as in the case of DB2 logging activity, and you have FWC (or ESS), the parity information for each set of chunks can be calculated without reading old parity data from the disk in many cases, therefore the effects of the write penalty are not significant. This is because all the required data (chunks) for the calculation of the parity information can be kept in the cache. Once a set of chunks has been put in the cache by the sequential write, the parity information is calculated and destaged to the disk storage with data chunks.

If you do not have FWC, mirrored disks would be a better choice for log files than a RAID 5 array.

3.2.2.5 What about RAID 10?

RAID 10 provides excellent performance with excellent availability. However, there is one main drawback with RAID 10, and that is cost. It suffers from the same disadvantage as RAID 1 (mirroring) in that the total number of disks required to hold the raw data will be doubled due to the fact that you are mirroring. For this reason, if overall cost is a concern, most people will opt for RAID 5; if cost is not a concern and you are using AIX 4.3.3 or later, then RAID 10 should be strongly considered where performance is the main goal. Tests have shown that RAID 10, or striping and mirroring as it is also known, provides close to the read/write performance of ordinary striped filesystems, with little additional overhead incurred due to the mirroring.

3.2.3 AIX logical volume parameters

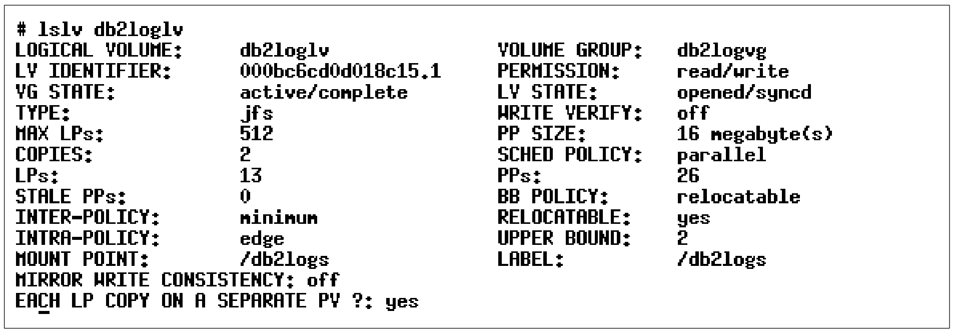
The AIX Logical Volume Manager (LVM) has many different components and features which affect performance. It would be impossible to cover them all in this book, but we will look at the main parameters for logical volumes which can influence the performance of your database. If you require detailed information on the AIX LVM then refer to the AIX Documentation or refer to the IBM Redbook Web site `http://www.redbooks.ibm.com` and search for the following books, you can view these on-line or order a hardcopy.

- *AIX Logical Volume Manager, From A to Z: Introduction and Concepts*, SG24-5432-00
- *AIX Logical Volume Manager from A to Z: Troubleshooting and Commands*, SG24-5433-00

3.2.3.1 Write scheduling policy

Earlier in this chapter we suggested that when using mirrored logical volumes the scheduling policy should be set to `Parallel`. The reason for this is that when performing a write operation with the `Sequential` policy enabled, the writes are performed sequentially, starting with the primary copy, and the write is not considered complete until the last copy is written. With the `Parallel` policy, however, all copies are treated equally and written in parallel so the write completion time is only constrained by the time it takes the slowest disk to perform the write.

To check the scheduling policy, you can use SMIT or the `lslv` command. The attribute to look for is `SCHED POLICY`, as shown in Figure 10.



```
# lslv db2loglv
LOGICAL VOLUME:     db2loglv                VOLUME GROUP:   db2logvg
LV IDENTIFIER:      000bc6cd0d018c15.1      PERMISSION:     read/write
VG STATE:           active/complete         LV STATE:       opened/syncd
TYPE:               jfs                     WRITE VERIFY:   off
MAX LPs:            512                     PP SIZE:        16 megabyte(s)
COPIES:             2                       SCHED POLICY:   parallel
LPs:                13                      PPs:            26
STALE PPs:          0                       BB POLICY:      relocatable
INTER-POLICY:       minimum                 RELOCATABLE:    yes
INTRA-POLICY:       edge                    UPPER BOUND:    2
MOUNT POINT:        /db2logs                LABEL:          /db2logs
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV ?: yes
```

Figure 10. lslv output

When creating a mirrored logical volume using the `mklv` command, the `-d p` flag and attribute will influence the scheduling policy.

In Figure 10, it can also be seen that we have WRITE VERIFY and MIRROR WRITE CONSISTENCY turned OFF. In both cases, this has been done to maximize the performance of the mirrored disks, but there are disadvantages to disabling each, which we will now look at.

3.2.3.2 Write verify

If WRITE VERIFY is turned ON, then for every write that is performed, a read will be done to verify that the write was successful. This obviously impacts the write performance of that logical volume. If set to OFF, then writes are not verified by the AIX LVM; this is the default setting.

3.2.3.3 Mirror write consistency

If MIRROR WRITE CONSISTENCY is turned ON, Mirror Write Consistency (MWC) will ensure that the data is consistent across ALL mirrored copies. If the system crashes or is not shut down properly, then MWC will identify which copies are inconsistent by marking them as "stale". When the system comes back up, if MWC is ON, then AIX will read the MWC log and make all copies consistent again.

From DB2's point of view, having MWC on for the log disks, for example, would prove very costly, as every write would consist of two operations, one to write the data and one to update the MWC cache record. Using hardware write cache such as that contained on certain SSA adapters can minimize this impact. If you do decide to disable MWC, then it will be necessary to put in place a procedure to re-synchronize the mirror copies in the event of a system crash. The volume groups containing such logical volumes would need to have their autovaryon capability disabled, and then after the system comes back up, each mirrored logical volume would need to be re-synchronized by running the following command:

```
syncvg -f -l <logical volume name>
```

If you are going to create such logical volumes and leave MWC set to ON, then write performance will be affected, so try to locate these logical volumes on the outer edges of the disks, as this is where the MWC cache is located. This will help minimize disk head movements when performing writes.

For more information on volume group attributes and selecting where on the disks to create logical volumes, refer to the *AIX Commands Reference*, SBOF-1851-00. Commands to look at are `mkvg` and `chvg` (for existing volume groups); `mklv` and `chlv` (for existing logical volumes).

Note

MWC will only ensure data consistency, NOT data integrity; this is the role of the database.

3.2.3.4 LVM_BUFCNT parameter

On AIX, when using applications that issue large raw I/O reads/writes which bypass the filesystem, as DB2 does when using DMS raw logical volumes, then we recommend that you increase the value for this parameter using the AIX `vmtune` command. The default for this is 9, which represents the number of LVM buffers allocated for raw physical I/O. In most cases, if you perform large physical I/Os to fast devices, then increasing this parameter should bring performance gains. This can be done using the `-u` flag of the AIX `vmtune` command.

3.2.4 SSA RAID array parameters

A common solution for database disk subsystems is RAID. On the AIX operating system, there are some tunable parameters which can improve the performance of your RAID array. We will now look at what these parameters are, how to check the current values you are using, and then how to change them if required.

3.2.4.1 Checking adapter / disk parameters

When using SSA RAID arrays, it is worth checking not only the attributes of the array itself, but also the extent and prefetch sizes you have used for table space containers defined on the array.

We will now use an example to demonstrate how to check these values and make recommendations on how SSA RAID array parameters can effect table space configuration decisions.

First we need to know what the RAID logical disks are on our system. To do this, we can run the following command — we can see here that on this system we only have one logical SSA RAID disk, hdisk4 (the logical disk will be made up of multiple physical disks).

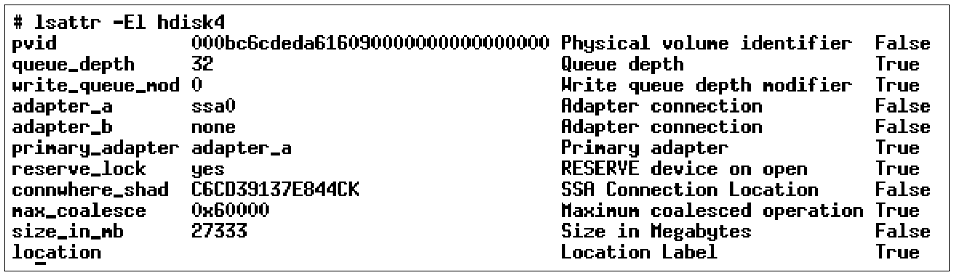
```
# lsdev -C -t hdisk -c disk -s ssar -H
name   status    location description
hdisk4 Available 10-70-L SSA Logical Disk Drive
```

To check the attributes we have set for this logical disk, we can use the command line as shown in Figure 11. The parameters that we are interested in checking are `max_coalesce` and `queue_depth`.

- **max_coalesce**
  Specifies the maximum number of bytes that the SSA disk device driver attempts to transfer to or from an SSA logical disk in one operation. For an N+P array, set this to 64K*N (0xN0000) (for example, 0x70000 for a 7+P array).

  Currently it is recommended that this value should not exceed 0xFF000, even though the SSA device driver allows values up to 2 MB. Check your SSA adapter documentation for the most up-to-date information.

- **queue_depth**
  Specifies the maximum number of commands that the SSA disk device driver dispatches for a single disk drive for an hdisk. For an N+P array, set to 2*N or even 3*N. By altering this value, we maximize the chance of reading data from each component of the array in parallel.



```
# lsattr -El hdisk4
pvid            000bc6cdeda616090000000000000000 Physical volume identifier  False
queue_depth     32                               Queue depth                 True
write_queue_mod 0                                Write queue depth modifier  True
adapter_a       ssa0                             Adapter connection          False
adapter_b       none                             Adapter connection          False
primary_adapter adapter_a                        Primary adapter             True
reserve_lock    yes                              RESERVE device on open      True
connwhere_shad  C6CD39137E844CK                  SSA Connection Location     False
max_coalesce    0x60000                          Maximum coalesced operation True
size_in_mb      27333                            Size in Megabytes           False
location                                         Location Label              True
```

Figure 11. SSA logical disk attributes

To set this value, use the following command for the logical SSA disk you want to change. The `chdev` command can also be used to alter `queue_depth`.

```
chdev -l hdisk# -a max_coalesce=(0x10000 * N)
```

Using Figure 11 as an example of a 3+P array, we would set the `max_coalesce` value to 0x30000 and `queue_depth` to 2 * N = 6.

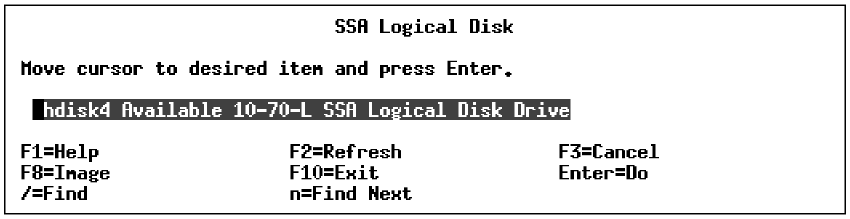
Note

Make sure any volume groups defined on the array are varied off before making changes to the above parameters.

This operation can also be performed using SMIT; the fast path for this is:

```
smitty chgssardsk
```

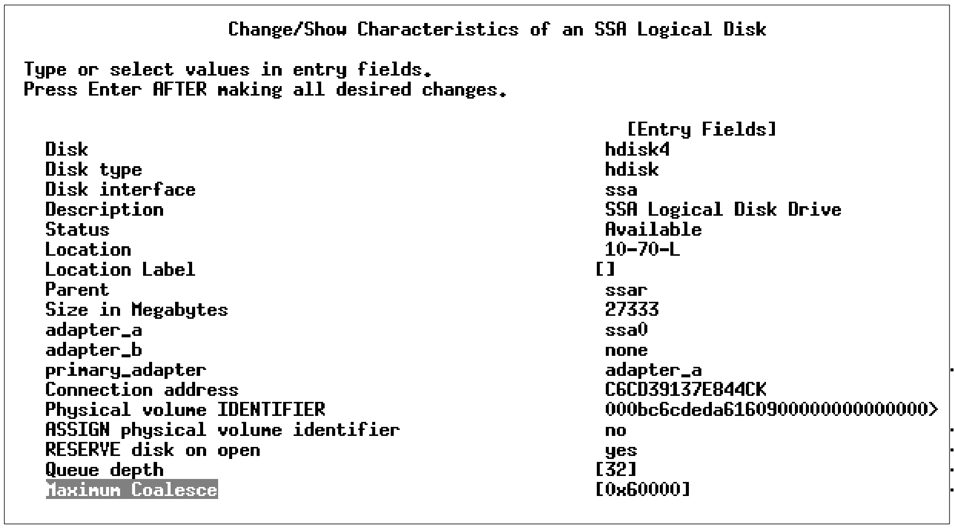
You will now be presented with a screen similar to Figure 12, from which you can select the SSA Logical Disk name:



```
                              SSA Logical Disk

Move cursor to desired item and press Enter.

  hdisk4 Available 10-70-L SSA Logical Disk Drive

F1=Help                 F2=Refresh              F3=Cancel
F8=Image                F10=Exit                Enter=Do
/=Find                  n=Find Next
```

Figure 12. Select SSA logical disk

We now see a screen as in Figure 13 below. Here you can see the `max_coalesce` and `queue_depth` settings:



```
                    Change/Show Characteristics of an SSA Logical Disk

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
  Disk                                                hdisk4
  Disk type                                           hdisk
  Disk interface                                      ssa
  Description                                         SSA Logical Disk Drive
  Status                                              Available
  Location                                            10-70-L
  Location Label                                     []
  Parent                                              ssar
  Size in Megabytes                                   27333
  adapter_a                                           ssa0
  adapter_b                                           none
  primary_adapter                                     adapter_a
  Connection address                                  C6CD39137E844CK
  Physical volume IDENTIFIER                          000bc6cdeda6160900000000000000>
  ASSIGN physical volume identifier                   no
  RESERVE disk on open                                yes
  Queue depth                                        [32]
  Maximum Coalesce                                   [0x60000]
```

Figure 13. Changing ssa logical disk attributes

As was stated earlier, you should set `max_coalesce` to 64 KB x N. The default is 0x20000, which corresponds to two 64 KB strips (and applies to a 2+P array).

For a 9+P array of 10 disks, set the value to 0x90000 which is 9x64KB.

For an 11+P array of 12 disks, this equates to 11x64KB (11 * 0x10000) = 0xb0000.

This is documented in the manual, *Advanced Serial RAID Adapters User Guide and Maintenance Information,* SA33-3285-01. This manual can be downloaded in PDF format from the URL:

`http://www.hursley.ibm.com/~ssa/docs/index.html`

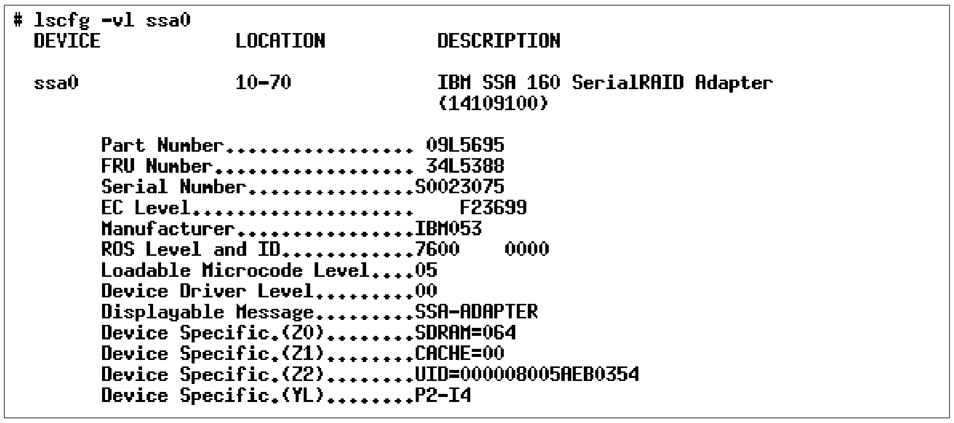
3.2.4.2 Fast write cache (FWC)

Fast write cache (FWC) is an optional nonvolatile cache that provides redundancy with the standard adapter cache. The FWC tracks writes that have not been committed to disk and can significantly improve the response time for write operations.

Fast write cache typically provides significant advantages in specialized workloads, for example, copying a database onto a new set of disks. If the fast write cache is spread over multiple adapters, this can multiply the benefit.

With SSA, the fast write cache can be used if you have a RAID array configured, or even if you are just using it as JBOD (just a bunch of disks).

To check if you have fast write cache modules installed, you will need to look at the vital product data (VPD) for your adapters. This can be checked using the SMIT `ssafastw` fast path, or from the command line as shown in Figure 14:



```
# lscfg -vl ssa0
  DEVICE            LOCATION          DESCRIPTION

  ssa0              10-70             IBM SSA 160 SerialRAID Adapter
                                      (14109100)

        Part Number................. 09L5695
        FRU Number.................. 34L5388
        Serial Number...............S0023075
        EC Level....................    F23699
        Manufacturer................IBM053
        ROS Level and ID............7600    0000
        Loadable Microcode Level....05
        Device Driver Level.........00
        Displayable Message.........SSA-ADAPTER
        Device Specific.(Z0)........SDRAM=064
        Device Specific.(Z1)........CACHE=00
        Device Specific.(Z2)........UID=000008005AEB0354
        Device Specific.(YL)........P2-I4
```

Figure 14. Using the lscfg command to show adapter information

Using the AIX command, `lscfg -vl <adapter_name>`, we can view the adapter information as shown above. From this we can see that the attribute Device Specific (Z1) tells us if the adapter contains a plugable fast write cache module; if we have such a module installed, then the number is the cache size in megabytes. As you can see in our example, there is NO fast write cache installed for this adapter.

You can also see, in Fig. 15 on page 47, how the `ssaraid` command can be used to show the status of the fast write cache attribute.

Note

The entry, Device Specific (Z0) represents the size, in megabytes, of the installed synchronous dynamic random access memory (SDRAM) modules. This is NOT fast write cache.

When enabling the fast write cache for a device, in order for the change to take effect, you must ensure that:

- All filesystems associated with the volume group the disk belongs to must be unmounted.
- Once filesystems are unmounted, the volume group must be varied off.

If you do have a fast write cache module installed, then to turn fast write cache on, you can use SMIT. The fast path command would be:

```
smitty ssafastw
```

From the command line, this can be done by running the following command:

```
ssaraid -H -l ssa0 -n hdisk4 -a fastwrite=yes
```

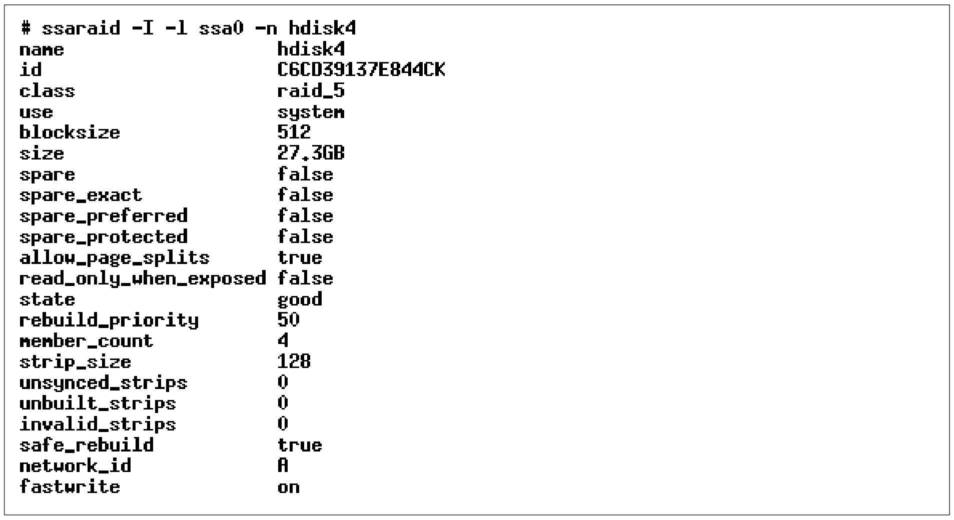
In this example, we are enabling the fast write cache for the SSA logical disk, hdisk4, which is attached to the ssa0 adapter. Once the change has been made, the volume group can be varied back on again, and the filesystems can be mounted.

3.2.5 Effects on table space configuration

When using SSA RAID devices, the physical disks will be striped at the hardware level beneath the database. If you are using DMS raw containers, then this striping will affect the values you use for extent size and prefetch size for the table space.

We would recommend that for any table space using a RAID device, you should define only one container. The reason is that, if you define multiple containers, then the data will be striped, not only across the DB2 containers, but effectively also striped at the hardware level by the RAID device. This gives us striping on top of striping, which is not desirable.

The SSA RAID logical disk attribute `strip_size` is the parameter that you need to look at. The `strip_size` parameter is set when the array is created. Once the array is created, you can query the value used for `strip_size` as shown in Figure 15; the value used for this parameter is in 512 byte blocks, **not** KB:



```
# ssaraid -I -l ssa0 -n hdisk4
name                    hdisk4
id                      C6CD39137E844CK
class                   raid_5
use                     system
blocksize               512
size                    27.3GB
spare                   false
spare_exact             false
spare_preferred         false
spare_protected         false
allow_page_splits       true
read_only_when_exposed false
state                   good
rebuild_priority        50
member_count            4
strip_size              128
unsynced_strips         0
unbuilt_strips          0
invalid_strips          0
safe_rebuild            true
network_id              A
fastwrite               on
```

Figure 15. Using ssaraid command to show strip_size attribute

In Figure 15, you can see that `strip_size` is set to 128 for this particular SSA RAID logical disk. This value represents 64 KB (128 * 512 byte blocks).

Because the SSA RAID array is striped in this way, it is recommended that you set the following DB2 registry variable BEFORE creating any table space containers on the array.

3.2.6 The DB2_STRIPED_CONTAINERS variable

A striped container is a DMS raw container that is mapped to a device which implements data striping beneath the database (at the file system or disk subsystem level). Set the variable as follows; this command would be run as the DB2 instance owner:

```
db2set DB2_STRIPED_CONTAINERS=ON
```

The reason we set this variable is that RAID does its own striping. It is important to ensure that DB2's containers line up with the RAID array's stripe units. To achieve this goal, we do two things:

- Use an extent size that is equal to, or a multiple of the RAID array's stripe unit. Remember that the stripe unit size returned by the `ssaraid` command is in 512 KB blocks. For example, when the `strip_size` returned by the `ssaraid` command is 128, the strip unit size is 64 KB, and it is recommended to use 64 KB or a multiple of 64 KB for the extent size.
- Set the `DB2_STRIPED_CONTAINERS=ON` registry variable. Because all DB2 containers contain a one-page tag, the extents will not line up with the RAID stripe units which could result in an increase in I/O requests. By setting this variable to ON and using DMS raw containers, each container tag is allocated one complete extent. This allows the stripe units and extents to line up correctly, but at the cost of a one-extent overhead. A `db2stop` and `db2start` must be done in order for this variable to take effect.

The reason this variable does not affect DMS file containers is that the pages allocated for the DMS file container are still allocated by the filesystem, so we cannot guarantee that the pages in the container will be contiguous across the disk(s). With raw devices, DB2 controls this and allocates the pages contiguously.

3.2.6.1 Setting up a table space on a RAID device

First we would follow the guidelines above for configuring the RAID array. We would then set the `DB2_STRIPED_CONTAINERS` registry variable to `ON` and follow that with a `db2stop` and `db2start`.

The next stage is to create the raw logical volume on the RAID array. Let us assume the logical volume is called `tbslv`. We now connect to the database and create the table space using a command like that shown below.

```
CREATE TABLESPACE TBS1
MANAGED BY DATABASE USING
(DEVICE '/dev/rtbs1lv' 2000) PAGESIZE 8K
EXTENTSIZE 8
PREFETCHSIZE 24
```

A relatively small RAID array configuration is used to illustrate these points:

Using an array = 3 + P (3 parallel plus one parity) and a strip size = 64k, we can say that the above `CREATE TABLESPACE` command will give us the following results:

- Using an `EXTENTSIZE` of 8 pages and a `PAGESIZE` of 8K, each extent will span 1 drive. The value used for `EXTENTSIZE` should always be equal to or a multiple of the RAID stripe size.
- Using a `PREFETCHSIZE` of 24 pages, each prefetch done will utilize the 3 parallel disks. The calculation is `strip size * N / PAGESIZE` which equates, in this example, to `64KB * 3 / 8 = 24`. We recommend that the `PREFETCHSIZE` is the RAID stripe size multiplied by the number of parallel disk drives (do not include parity disk) and also a multiple of the `EXTENTSIZE`. If you wish to increase the `PREFETCHSIZE`, then you also need to consider increasing the `NUM_IOSERVERS` database configuration parameter to a least `PREFETCHSIZE / EXTENTSIZE`.

If the database's main workload requires good sequential I/O performance, such as a DSS workload, then the `PREFETCHSIZE` becomes even more important.

In our example, we have been using DMS raw containers. When this type of container is used, the operating system does not do ANY prefetching or caching, so ensure that you use an adequate `PREFECTHSIZE` and enable the table space for parallel I/O.

3.2.7 The DB2_PARALLEL_IO variable

So far, we have looked at the use of the DB2 registry variable `DB2_STRIPED_CONTAINERS` and its use with containers defined on RAID devices.

Another important registry variable is `DB2_PARALLEL_IO`. We mentioned earlier that, for table spaces that exist on RAID devices, you should only create one container. Obviously, the array itself is made up of more than one physical disk, so it is recommended that for this type of table space, you should enable this registry variable. When you specify the table space's ID to the `DB2_PARALLEL_IO` registry variable, the database is able to issue parallel I/O requests to that table space, even though it consists of a single container. DB2 achieves this by enabling multiple prefetchers simultaneously for the table space which results in improved utilization of the underlying physical drives.

You can enable parallel I/O for every table spaces using the `DB2_PARALLEL_IO` variable, as shown in the following example:

```
db2set DB2_PARALLEL_IO=*
```

You can also enable parallel I/O for a list of table spaces IDs (comma-separated) as in the following example:

```
db2set DB2_PARALLEL_IO=1,2,4
```

A list of table space IDs can be obtained by running the `LIST TABLESPACES` command when connected to your database, or you could run this command:

```
SELECT TBSPACE,TBSPACEID FROM SYSCAT.TABLESPACES
```

After setting the `DB2_PARALLEL_IO` registry variable, you must run `db2stop` and then `db2start` for the change to take effect.

`DB2_PARALLEL_IO` should also be enabled for striped containers when `PREFETCHSIZE > EXTENTSIZE`

Note

Also refer to the table space parameters `OVERHEAD` and `TRANSFERRATE` discussed in 3.3, "Table spaces: performance considerations" on page 52.

3.2.8 Multi-page file allocation

With SMS table spaces, when you are inserting or updating a table in the table space, the file associated with the SMS container is extended one page at a time, which can lead to a performance degradation if you are inserting large numbers of rows.

For SMS, turn on `multipage_alloc` to improve insert performance; note that this is a database configuration parameter that controls whether pages are allocated by page (default) or by extent. To set this, you must use the `db2empfa` tool to turn `multipage_alloc ON`, as follows:

```
db2empfa <database-alias>
```

This parameter has NO effect on DMS table spaces (as space is pre-allocated) or temporary table spaces (as all required pages are allocated when temporary tables are created).

Note

Once this parameter has been set to YES for a particular database, it cannot be set back to NO.

3.2.9 Using Enterprise Storage Server

With the introduction of the Enterprise Storage Server (ESS), it is possible that this may be the main component of some DB2 disk subsystems.

If you are using such hardware, then you may be wondering if there is anything specific you need to look at from a DB2 configuration point of view.

One recommendation would be to look at the value being used for the `NUM_IOSERVERS` parameter. We would recommend that a good starting value would be to set the `NUM_IOSERVERS` equal to the number of the ESS's ranks.

Note

One ESS rank is a set of 7 Raid-5 disks (a 6+P array). Normally you will have two spare disks per SSA loop, so the array may be known as a 6+P+S. In ESS the spare disk is able to "float" between different arrays.

Using that as a starting point, `NUM_IOSERVERS` can then be increased, but some tests have shown that, in some cases, this does not provide any performance improvements; in fact, there may be some small overhead in going for a larger value. This will ultimately be up to the database administrator to test, as all environments are different, and the value used will depend on factors such as workload type, for example, DSS or OLTP.

`DB2_PARALLEL_IO` and `DB2_STRIPED_CONTAINERS` should also be set when using ESS units.

If you use your ESS unit, which has a large cache, with a system which also has a large amount of physical memory dedicated to DB2 (such as large buffer pools) then the `NUM_IOCLEANERS` value need not be very large. Using the number of CPUs in the system would be a good starting point.

It is also recommended that if your database uses mainly SMS table spaces, then having more containers per ranks of ESS gives better performance.

For example, suppose you create a filesystem on one ESS rank, then define it as a container for one of your SMS table spaces. Doing this could cause a performance degradation due to i-node lock contention in the operating system. By defining multiple containers for the SMS table space, you would reduce the i-node lock contention. Some tests have shown that allocating four containers per ESS RAID array achieves very good results.

3.2.9.1 DMS considerations

In an ESS system, each RAID array (ESS ranks) can be defined as one logical disk or multiple logical disks. AIX sees each logical disk as an hdisk. When creating logical volumes in AIX for use as DMS device containers, the AIX Logical Volume Manager (LVM) will allow you to spread the logical volume(s) across multiple hdisks, and in a non-RAID environment, we would normally do this. However, if you are spreading the logical volume across an array which you have split up into multiple logical disks, then you will not get any performance advantage over a setup where you had defined just one logical disk for the array. The reason for this is that all I/O activity still goes to the same RAID array.

Normally, to maximize the throughput of an ESS system, you would balance your workload across as many ESS ranks (RAID5 arrays) as possible. By defining logical volumes across multiple ESS logical disks on separate RAID5 arrays, you could achieve this. However, DB2 attempts to balance load across table space containers, so if each container is a DMS device container, then try to define one container per RAID5 array. For this configuration we would require one logical disk (AIX hdisk) per array. By having one logical disk per array, we would not need to use the AIX LVM to create the logical volume across multiple disks in the same array.

3.3 Table spaces: performance considerations

Table spaces in DB2 can be defined as either of the following two types, which we summarize in the following sections.

3.3.1 SMS table spaces

Containers are operating system directories; you can easily increase table space capacity by enlarging the underlying operating system file system. Data is striped across the container by extent; disk space is allocated on demand one page at a time (by default).

With SMS table spaces, data objects (table data, indexes, LONG VARCHARs, and LOBs) are located by operating system file name.

We suggest that each container (directory) be associated with a different file system; otherwise, table space capacity will be limited to that of a single file system.

Ensure that containers have equal capacity, as excess capacity in larger containers is not exploited.

These are some advantages of using SMS table spaces:

- Space is not allocated until required.
- Initial database creation may be simpler (no DMS container definitions required).

3.3.2 DMS table spaces

Containers are either operating system files or raw devices. We recommend that, where possible, you should associate each container with a different disk or disks, as this enables parallel I/O, and gives the opportunity for larger table space capacity.

With DMS, you can increase table space capacity via `ALTER TABLESPACE ADD CONTAINER`, or use a new feature in DB2 v7.1 — the addition of two new clauses for the `ALTER TABLESPACE` statement. These are `RESIZE` and `EXTEND`. These options will be discussed in 3.3.9.2, “Container sizes” on page 68.

When using DMS device containers in a table space, you need to understand the following performance considerations:

- File system caching for DMS file containers is performed by the operating system in the filesystem cache; on AIX this is the JFS cache.
- File system caching for DMS device containers (raw logical volumes for example) is NOT done by the operating system.

In DMS table spaces data, is striped across the container(s) by extent, as in Figure 16. Note that extents are mapped in a round-robin fashion, and extents for different objects (table 1 and table 2) need not be contiguous.

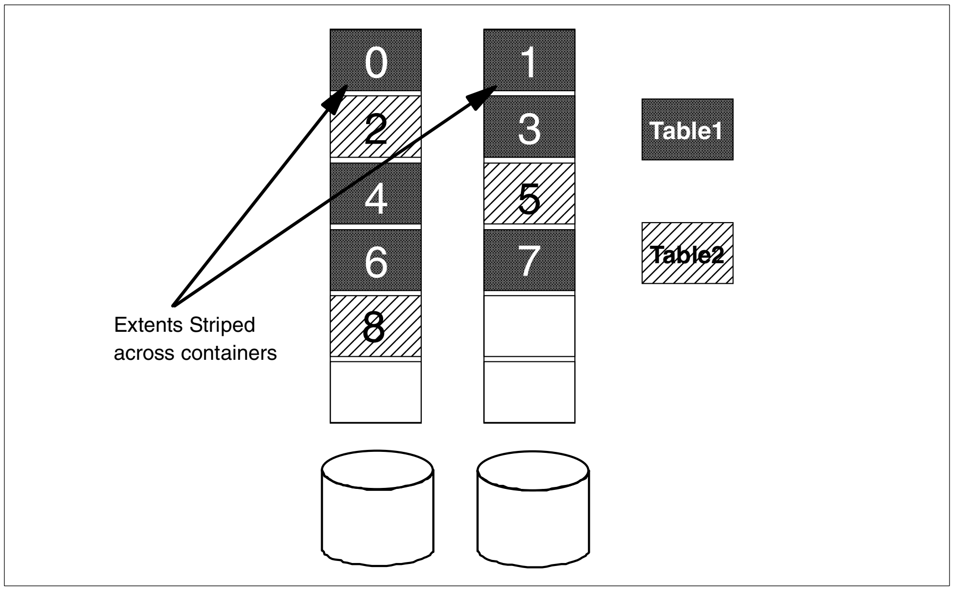


Figure 16. DMS table space structure

Here are some advantages of using DMS table spaces:

- Containers can be added and extended.
- Large tables can be split up by data type (LOBs, indexes, data) across multiple table spaces.
- DMS device placements on disk can be controlled using the logical volume manager (outer edge, middle)
- DMS, in most situations, will give better performance than SMS.

3.3.3 SMS versus DMS

In some situations, it can be difficult to decide whether to use SMS table spaces or DMS table spaces, as there are a number of different factors to consider. We strongly recommend that, where possible, you should use DMS table spaces with device (raw logical volume) containers if performance is your main priority. The database can do a much better job than the filesystem when it comes to managing its own disk blocks. However, in this section of the chapter we will discuss situations where either SMS or DMS file containers may be a reasonable and sometimes more suitable alternative.

When using DMS raw devices, DB2 will ensure that pages are placed contiguously on the disk drives; with SMS containers, this is not the case, as the filesystem decides where to locate the pages (this can also apply to DMS file containers). Contiguous placement of pages is important, as pages make up DB2 extents. Contiguous pages will speed up operations like table scans.

3.3.4 Table space categories

In DB2 we have seen that table spaces can be either SMS or DMS. We can also break down table spaces into the following categories which represent the data type for a table space.

3.3.4.1 Regular table spaces

You can think of a regular table space as a table space for containing user data; the default table space for this data type is called USERSPACE1. Index data is also stored in regular table spaces, as are the system catalog tables. The default table space for system catalogs is called SYSCATSPACE. Both USERSPACE1 and SYSCATSPACE are created when the CREATE DATABASE command is run, and both are created as SMS type table spaces by default.

You can, of course, drop USERSPACE1 if you wish, as its use for data is optional. Mostly, you will create additional table spaces for your data. However, the name SYSCATSPACE **must** be used for the table space which is holding the system catalogs.

3.3.4.2 Long table spaces

Any tables which contain long field data or long object data will occupy this type of table space.

When any application has a need to access data from a long table space (whether the data be LOBs, LONG VARCHAR, or LONG VARGRAPHIC), the database manager cannot use the buffer pools to cache the data. Therefore, every time a page is requested, the database manager must request it from disk (see point in this section on DMS file containers to help with this).

Long table spaces must be DMS type table spaces. Use of long table spaces is optional, as the data type can reside in regular table spaces.

From a performance point of view, as LONG data is NOT stored in the buffer pools, then you can use DMS file containers instead of raw devices. The reason for this is that by using the file containers, you will benefit from the operating system's filesystem cache.

3.3.4.3 System temporary table spaces

System temporary table spaces are used during SQL operations for internal temporary tables, for sorts, to store intermediate results, table reorganizations, and other transient data.

All databases in DB2 must have at least one system temporary table space. The default, called TEMPSPACE1, is created by default as an SMS table space.

3.3.4.4 User temporary table spaces

Introduced with DB2 v7.1 is a new table space type called user temporary table spaces. This table space type was introduced to support another new feature in this release of DB2 which is discussed in 4.1.3.6, "Declared temporary tables" on page 101. Therefore, the main use of this table space type is to store the declared temporary tables once a user or application has defined such a table.

3.3.5 Choosing table space types for data tables

When deciding what type of table space to create to provide optimal performance (for example, to maximize I/O throughput), you need to be familiar with the following concepts before making your decision:

- Big block reads, a read where several pages (normally an extent are retrieved in a single request.
- Prefetching, reading pages in advance, in an attempt to reduce query response times.

- Page cleaning, the writing of modified buffer pool pages to disk to make room for a new page to be read in.

DB2 tries to perform big-block reads whenever possible to maximize the amount of data read in one operation. When using DMS device containers the data will in most cases be contiguous on the disk, which reduces seek time and latency. When using SMS or DMS file containers, then the operating system's filesystem may have broken the files up and stored the different parts of the file in different locations on the disk(s) which means the file has become fragmented. This obviously gives a read performance degradation.

Prefetching is discussed later in this chapter.

3.3.5.1 Data tables

Raw data and indexes are classed as regular user data. To maximize performance, choose DMS with raw containers. SMS has always been thought of as giving a lower level of performance but superior (simpler) system administration. However, due to a number of enhancements to DMS with DB2 v7.1, this argument is weakening slightly (features like the DMS table space resize option affect this).

As we have said already, raw containers will, in nearly all cases, outperform file containers because the database manager can bypass the operating system's file system and also avoid unnecessary double buffering.

Again, `LOB` or `LONG VARCHAR` data benefits from the use SMS or DMS table spaces with FILE containers, as they benefit from the operating system's filesystem cache. `LOB` and `LONG VARCHAR` are not buffered in DB2's buffer pools, as they use direct I/O. Any memory allocated to long table spaces should be kept to a minimum, or alternatively, just use the default buffer pool `IBMDEFAULTBP`.

3.3.5.2 Temporary table spaces

We suggest that for nearly all systems, SMS is the right table space type to choose for both system and user temporary table spaces.

The main advantage of using SMS for temporary table spaces is that data stored in temporary tables is mostly transient data (for example batch work). By using SMS you will only allocate disk space as and when required, unlike DMS, which will allocate all specified space when the table space is created.

If you have table spaces utilizing different page sizes, then our suggestion would be to create one temporary table space with a page size equal to that of the majority of your data tables (see note on reorg). When selecting temporary table spaces, DB2 ignores page sizes which are too small to accommodate the operation which requires temporary space. For this reason you will need to ensure that the page size of your temporary table space(s) can accommodate the row lengths and column used in your database; see Table 2 on page 64 for a list of page size row and column limits.

The reason we suggest only having one temporary table space for a given page size is because when you have multiple temporary table spaces using same page sizes, the DB2 optimizer will generally choose a temporary table space size by selecting the temporary table space with the largest buffer pool. Let us say you had two 16 KB temporary table spaces defined called `TEMP1` and `TEMP2`. Now assume `TEMP1` has a large buffer pool and `TEMP2` has only a small buffer pool; UDB will select `TEMP1` as it has largest buffer pool. Once selection is made, UDB will then alternate between ALL temporary table spaces which have the chosen page size (16 KB).

This means that when performing an operation such as a sort, we alternate between `TEMP1` and `TEMP2` because they both use a 16 KB page size. The disadvantage here is that we only use `TEMP1`'s large buffer pool half the time, because we are alternating between the two table spaces. A better solution in this example would be to have a single 16 KB temporary table space with one buffer pool.

Note

If you reorganize table spaces in the temporary table space, then you **will** need a temporary table space which has the **same** page size as the table space being reorganized.

3.3.5.3 Catalog table spaces

The system catalogs contain some LOB columns, as we described earlier in this section. When the database manager needs to retrieve a page of LOB data, it cannot use the buffer pools as a cache. The only way to improve performance by providing some buffering is to use SMS or DMS file containers. When these container types are utilized, the operating system's filesystem cache will provide some buffering.

For the catalog table space, use SMS or DMS with FILE containers and a small extent size (2 or 4 pages).

The reason for this is that the catalogs contain a lot of relatively small tables. DMS requires 2 overhead extents per table (Extent Map Page + 1st data extent). SMS requires only 1 page.

3.3.6 Deciding number of tables and table spaces

A common question asked is, "How many tables should I put in a table space?". This in turn affects another frequently asked design question, "How many table spaces should I create?" Whatever number is used, the decision can affect the performance of that particular table and table space.

Here are some considerations for placing tables, which might help you decide which tables should go:

- When recovering your table spaces, if you have a collection of tables related by Referential Integrity (RI) constraints or summary tables, then if you ever need to restore the tables, they should be rolled forward together to a point in time. If this is not done, then one table, or more, will be placed in a check pending state.
- If you are using DMS table spaces, you then have the option of creating what are known as "spanned" tables. A spanned table has data pages in more than one table space. The best example of this is a table which has its data in one table space, indexes in another, and LOB data in another. Only DMS supports this. An advantage of using spanned tables is that, by separating data and indexes, you can place the indexes in a table space with its own dedicated buffer pool, which can help ensure that index pages are kept in memory.
- How much raw data is in the tables is also important. If you have a number of small tables with a small amount of data which are not changed frequently, then it is feasible to group these tables together in a single table space. Larger base tables which are heavily accessed and frequently updated can justify having their own DMS table space. From a performance point of view, if a very large table has its own table space, then you can also assign it a dedicated buffer pool (see 3.4.1.2, “Multiple buffer pools” on page 72 for details on this subject). By grouping tables in specific table spaces, you can perform flexible backup strategies as you have more granularity.
- Keeping important tables in their own table space will simplify administration of this table space and will speed recovery should the table space need to be restored (because you will only have one table to restore). When required, you can also create a dedicated buffer pool for these tables.
- Group infrequently accessed data in table spaces which use containers on slower devices.
- Minimize relationships (such as RI constraints) between table spaces; this simplifies recovery.

- Consider table space capacity limits: 64 GB (4K pages), 128 GB (8K pages), 512 GB (32K pages), 2 TB for long table spaces. These are the limits for DB2 UDB Enterprise Edition (EE). If you use DB2 UDB Enterprise-Extended Edition (EEE), they are the limits per partition.
- Consider leaving room for in-place table reorganization, which is faster than reorganization using a system temporary table space because data copied only once and not copied back and forth between temporary table space and source table space.
- Place tables that need to be monitored more in their own table spaces, since the `LIST TABLESPACE SHOW DETAIL` command gives the number of used pages. If the table space is a DMS table space, the command also reports the number of free pages and the high water mark information. For example, you can monitor the growth of a fact table of a star schema by putting it in its own table space.
- Try to minimize the overall number of table spaces.

3.3.6.1 Recoverability of "related" table spaces

It is worth remembering when assigning tables to table spaces that if you ever need to recover a table space, then "relationships" with tables in other table spaces will be a factor. For example, assume we have two table spaces TS1 and TS2; we recover TS1 from a backup, and then wish to roll forward to a point in time. We then realize that one of the tables within T1 is a summary table for one of the tables in T2. If we now simply roll forward TS1 and do not roll forward TS2 to the same point in time, the summary table in TS1 will be placed into a check pending state. The same would apply if the table in TS1 was the underlying table for the summary table contained in TS2.

3.3.7 Choosing table space containers

A container in DB2 is a physical storage device, and each container is assigned to one table space. Containers can be either files, directories, or devices; we will look at these in this section. Whichever container type you decide to use for a table space, you should stay with it. You will NOT gain good performance by mixing container types within a table space.

Over the last few years, many people have performed benchmarks on device containers (raw logical volumes) versus directory and file containers (JFS file systems). In nearly all cases these benchmarks have shown an overall improvement in disk I/O throughput of 10-35 percent when using device containers, as compared to JFS file systems. However, the database administrator needs to take into account that actual gains will vary, depending on the I/O workload mix of the application, OLTP or DSS.

Workloads like OLTP, which perform a large amount of random I/O operations, benefit from the use of raw logical volumes. Customers with DSS applications, which perform a large amount of sequential I/O operations, may generally benefit from the sequential read-ahead feature of JFS file systems. Even in this DSS environment, we still would favor DMS device containers, which allow DB2 to ensure that pages are placed contiguously (this influences things like prefetch performance).

3.3.7.1 Directory

This is the only container type that can be used in SMS table spaces. When creating SMS table spaces, make sure you specify at creation time all the directories (containers) that you want to incorporate into the table space; additional containers CANNOT be added once the table space exists.

To improve performance with this container type, we recommend that each container be mapped to a different physical drive to improve parallel I/O.

3.3.7.2 File

File containers are used by DMS table spaces and are files of a pre-allocated size. DMS treats file and device containers in the same way. When choosing a size for the file, be aware that the space required, for the chosen size, is pre-allocated. When the file is created, DB2 will allocate the entire container (file) at table space creation time. Even though this allocation is done at creation time, the operating system's filesystem may still fragment the file, so in some cases, the allocation of pages may not be contiguous.

If using DMS table spaces, then in most cases, file containers will give poorer performance when compared to raw device containers. The reason for this is that with file containers, the database manager has to interact with the operating system's filesystem layer, unlike raw device containers, which the database manager manages directly. An exception to this rule would be a DMS table space using file containers that was created to hold LOB data. In this case, performance might be greater than that of device container, since DB2 can take advantage of the filesystem cache. Remember that LOB data is NOT buffered by DB2, so using device containers would result in direct disk I/O operations when accessing LOB data.

3.3.7.3 Device

On AIX, device containers are commonly known as raw logical volumes. This type of container can only be used by DMS table spaces. When using device containers, the space, as with file containers, is also allocated at table space creation time. However, in this case, DB2 interacts with the raw device directly, which ensures that page allocation is contiguous.

The recommendation would be to use device containers when maximum performance is your goal, the exception being DMS table spaces designed for LOB data, as mentioned earlier.

The DMS and device container combination avoid the double buffering scenario which can occur with DMS file containers or SMS directory containers. This occurs when pages are cached by the operating system in its filesystem cache and by DB2 in the buffer pools. This scenario is a waste of resources.

3.3.8 Configuring table space containers

Before you create your table spaces, you need to think about how the container configuration will affect the overall performance of the table space. We will now look at the major factors affecting table space container configuration.

3.3.8.1 Disk considerations

The total number of disks is determined by the amount of raw data you have to store. For performance, the main thing to remember is that by using multiple disks for your containers, you will enable DB2 to perform parallel I/O. For example, you would generally achieve better performance with a large number of low-capacity disk drives, allowing you to have a drive for each table space container, than you would from having a small number of high-capacity disk drives with multiple containers on each drive.

For random access workloads (such as OLTP), you will need as many disks in the table space as are necessary to support the required I/O. For this type of workload, writes will be the main concern, as they are more expensive than reads (RAID systems), therefore the read/write ratio of one physical write has to be considered.

For sequential access workloads (such as DSS), some of the same points apply; you will need sufficient disks to support the required I/O. However, in most situations, DSS workloads will be more “disk-friendly”; this is because they have larger I/O sizes, which are generally sequential. This, along with other factors such as prefetching and disk caching, means that a DSS workload may be able to work efficiently with fewer disks than an OLTP system, with similar I/O requirements.

Other disk recommendations would be:

- Try to spread your table spaces over as many disks as possible; however, do NOT place more than one heavily accessed table space on a disk.
- If you only have a small number of disks available, then it is better to spread ALL the table spaces over ALL the disks rather than to assign only one disk to each table space.

There are also two table space configuration parameters that can be set with either the `CREATE TABLESPACE` or `ALTER TABLESPACE` statements, which will also affect the performance of your table space containers. These are defined in two columns contained in `SYSCAT.TABLESPACES`, and the values can be shown by running the command:

```
SELECT TBSPACE,OVERHEAD,TRANSFERRATE FROM SYSCAT.TABLESPACES
```

The DB2 optimizer considers these two parameters when determining the I/O cost of accessing data from a particular table space. The results the optimizer obtains can affect a number of its decisions, for example, whether or not to use a particular index contained in a certain table space container, or which table to select for the inner and outer tables in a join statement.

Overhead

Overhead is an estimate (in milliseconds) of time required by the container before data is read into memory. The following formula can be used to estimate overhead cost (I/O controller, latency, seek time). A detailed explanation of this formula can be obtained in the *DB2 UDB Administration Guide: Performance,* SC09-2945.

```
OVERHEAD = average seek time in milliseconds + ( 0.5 * rotational latency )
```

In this formula, 0.5 represents an average overhead of one half rotation and rotational latency (milliseconds / full rotation) is given as:

```
( 1 / RPM ) * 60 * 1000
```

So, for a disk with an RPM = 10000, we can obtain:

```
( 1 / 10000 ) * 60 * 1000 = 6.0 milliseconds
```

So, assume an average seek time of 5.4 milliseconds for this disk type; this gives the following estimated OVERHEAD:

```
5.4 + ( 0.5 * 6.0 ) = 8.4 milliseconds
```

Transfer rate

This is also an estimate given in milliseconds. `TRANSFERRATE` represents time taken to read one data page into memory.

Assuming your table space containers represent one single physical disk, then the *DB2 UDB Administration Guide: Performance*, SC09-2945 provides the following formula:

```
( 1 / spec_rate ) * 1000 / 1024000 * PAGESIZE
```

In this formula, spec_rate is disk specification for transfer rate (MB/sec) and PAGESIZE represents the table space PAGESIZE value you have chosen for the table space which owns this container.

DB2 uses default values of 24.1 milliseconds for OVERHEAD and 0.9 milliseconds for TRANSFERRATE.

When calculating the TRANSFERRATE you need to take into account the following considerations if your table space containers are made up of more than one single physical disk (for example, an array such as RAID):

- If the spec_rate value is small, then multiply by the number of physical disks on which the container resides, assuming bottleneck at disk level.
- If the number of disks is large, then I/O bottleneck is not likely to be due to disks, but more likely due to disk controller or I/O buses (system). If this is the case, then you will need to test the actual I/O rate. Refer to the *DB2 UDB Administration Guide: Performance*, SC09-2945 for further suggestions on how to do this. One suggestion would be to monitor the sequential scan of a large table.

We recommend that if your container spans multiple physical disks, then these disks should ideally have the same OVERHEAD and TRANSFFERATE characteristics.

If they are not the same, or you do not have the hardware documents that contain the information you require for the formulas given, then set the OVERHEAD value to the average of all the disks in the table space. As for TRANSFERRATE, if no figures are available, try to set this to the value for a single physical disk for OLTP environments (or the value of the slowest disk). For a DSS environment, set TRANSFERRATE to the sum of all the disks. If your workload is a mixture, then use a figure in between the recommendations for OLTP and DSS.

3.3.8.2 Page size

The overall performance of your table space will be affected by the page size you choose when issuing the CREATE TABLESPACE statement. The type of workload you are going to put on the tables within the table space will help determine an optimal page size.

If your applications perform mainly random reads/writes, then we would recommend that a small page size should be used, as this will not waste space in the buffer pools. If, however, your workload is more DSS than OLTP, and accesses rows sequentially, then larger page sizes will provide better performance.

Note

Use large page sizes for temporary table spaces, as rows in tables that reside in this type of table space will be accessed sequentially.

However, there is an exception to this rule. When row size is smaller than page size/255, we will have wasted space on each page, as there is a maximum of 255 rows per page. If this applies to your system, then using a smaller page size may be more appropriate. Remember that the database manager allocates 76 bytes in each page (regardless of size) for control information. So using the default size of 4 KB as an example, this would leave 4020 bytes for data. Maximum row lengths and column limits for each page size are shown in Table 2.

Larger page sizes may allow you to reduce the number of levels in the index, since more index keys will fit on each index leaf page. However, remember that each index page can have only 255 index entries. Therefore, if your index key size is small, a good percentage of the page might be wasted with a large page size.

Table 2. Page sizes, row, and column limits

Page size	Maximum row length	Maximum columns
4 KB	4,005 bytes	500
8 KB	8,101 bytes	1012
16 KB	16,293 bytes	1012
32 KB	32,677 bytes	1012

Note

If you use multiple page sizes in your database, and you plan to reorganize your tables using the temporary table space (by specifying the `USE` option of the `REORG TABLE` command), you need to ensure that you have a temporary table space with the same page size as the table.

3.3.8.3 Extent size

The extent size you choose for a table space determines the number of table data pages written to one container before DB2 writes to the next container; the type of table space used is a major factor. When using DMS table spaces, we allocate space one extent at a time; when one extent is full, a whole new extent is allocated. This differs from SMS table spaces which allocate space one page at a time.

This allocation policy will influence your table space performance. Let us assume you have a number of small tables which you want to group together in one table space. If you decide to use DMS, when a new table is created, two extents are immediately required for the table object. If you have many small tables in this table space, then the database manager would allocate a lot of disk space to store a small amount of data.

In this situation, the recommendation would be to use a small extent size, or if performance is not critical for these tables, then you can use an SMS table space which allocates pages one at a time.

You may notice when using DMS table spaces that a number of pages are used up when you first create the table space and then create the first object (table). Figure 17 shows how these pages are used up on the initial creation of both the table space itself and the object. Note that each subsequent table added to the same table space will incur a minimum overhead of two extents.

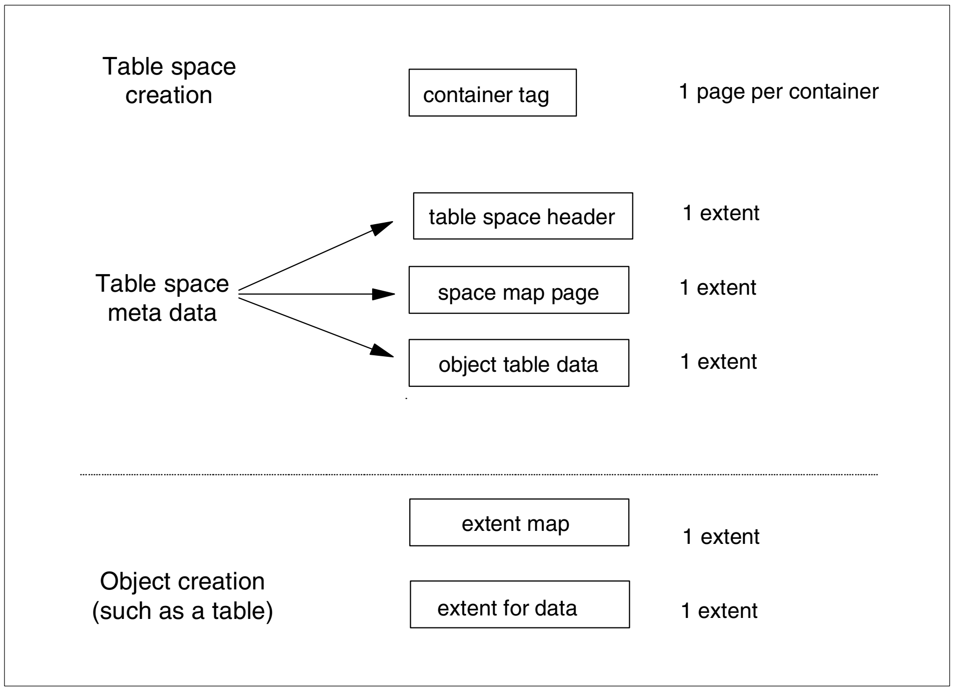


Figure 17. Minimum initial extent requirements for DMS table space

When the first storage object in table space is created, the following pages will be required, as shown above (assuming `EXTENTSIZE=32`):

Space required = 5 extents + 1 page = (5 * 32) + 1 page = 161 pages

The default `EXTENTSIZE` of 32 pages is generally acceptable for most systems, but this should be reduced if your table space consists of many small tables. Use larger values for tables that are mostly scanned (query only) or have a very high growth rate. If your DMS table space does contain large tables, and you do not increase the value for `EXTENTSIZE` but instead use a small extent size, then you will incur an overhead when the database has to continually allocate these small extents (either when a table is expanding or new tables are being created).

When using "striped" containers (like RAID), extent size should be set to a multiple of the stripe size (such as RAID strip size), as we discussed in 3.2, “Mirroring, striping, and using RAID devices” on page 35.

3.3.8.4 Prefetch size

Setting the prefetch size correctly will improve the database manager's ability to read pages in advance and reduce execution times. The default value for prefetch is defined by the database configuration parameter `DFT_PREFETCH_SZ`. Most customers, however, change this value using the `ALTER TABLESPACE` command, or they set it explicitly when they run the `CREATE TABLESPACE` command.

For most environments, the recommendation would be to set the `PREFETCHSIZE` equal to (`EXTENTSIZE` * number of containers). For optimal parallel prefetching, the containers in your table space should reside on separate physical devices. This is important when using a DMS raw container, as there will be no prefetching or caching performed by the operating system.

If you want to perform aggressive prefetching initially, try doubling the figure obtained after using the formula (`EXTENTSIZE` * number of containers). Regardless of whether you want to increase or decrease prefetching, it is best if the value used is still a multiple of `EXTENTSIZE`.

For `PREFETCHSIZE` considerations with RAID devices, see 3.2, "Mirroring, striping, and using RAID devices" on page 35.

The `NUM_IOSERVERS` database configuration parameter specifies the number of the I/O servers which perform prefetch I/O for the database. You should normally set the same number for the `NUM_IOSERVERS` parameter as the number of physical disks. If `PREFETCHSIZE` is increased above the suggestions outlined so far (for "additional" prefetching), then the `NUM_IOSERVERS` database configuration parameter should be set to at least `PREFETCHSIZE /EXTENTSIZE`.

Note

To disable prefetching on a table space, set `PREFETCHSIZE` to 0 using the `ALTER TABLESPACE` statement.

3.3.9 Deciding how many containers to use

Once the decision has been made as to the configuration of the table space containers, the next set of questions that commonly arise are to do with the number of containers to use in a table space and what factors affect this decision.

3.3.9.1 SIngle or multiple containers

Apart from the considerations we will look at next in 3.3.9.2, “Container sizes” on page 68, there are some other points to be made regarding having single or multiple containers in a table space and how this affects performance.

As long as the number of table spaces is small, then it is possible to have a one-to-one mapping of table space containers to physical volumes. This greatly simplifies the conceptual view of the physical database.

Normally you want to map each disk (or filesystem) to its own container for the following reasons:

- Isolation from disk errors
- Ability to better isolate parts of a workload (or database) from one another:

  For example, separating tables scanned simultaneously into separate table spaces with separate disks can reduce disk head movement (thus seek time) when doing sequential scans.

  As another example, you may want to protect time-critical data from random/adhoc queries on other data (remember that there are usually still shared CPUs and buses, so this is not always effective).

3.3.9.2 Container sizes

One of the factors which will affect the parallel I/O performance is the size of the containers used. If you use multiple containers of different sizes in a table space, then the effectiveness of any parallel prefetches performed against that table space will be reduced, for example:

- You create a new table space containing one small container and one large container. Eventually the small container will become full and any additional data will be written to the large container. The two containers are now unbalanced. Having unbalanced containers reduces the performance of parallel prefetching to nothing, as a `SELECT` statement may result in some data being required from the small container, but a large amount may be required from the large container.
- You have a table space which contains a number of large containers; these start to fill up, so you add a small container. The table space will then be rebalanced and the small container will contain far less data than will now reside in the larger containers, and once again the database manager will not be able to optimize parallel prefetching.

In summary, if you must use multiple containers, keep them the same size for best performance, and as we discussed in 3.3.7, “Choosing table space containers” on page 59, keep the container types the same if performance is your goal.

Note

Introduced with DB2 UDB v7.1 is the ability to increase the size of DMS containers using the `RESIZE` or `EXTEND` clauses of the `ALTER TABLESPACE` command. See the *DB2 UDB SQL Reference*, SC09-2974 for more detailed information.

3.3.9.3 Factors affecting number of containers

When you decide the number of containers, you should consider the following factors:

Availability

Other than parallel I/O performance, the main advantage of mapping one container to each physical disk is availability. With this system setup, disk errors will be isolated to only one container and therefore only one table space.

Load

With disk drive capacity increasing all the time, it may not be cost effective for you to have only one container per disk (especially as the disk could be 18 GB, 32 GB, or more). However, there is a scenario in which having more than one container on a disk may not cause you any performance problems. Imagine that you have two table space containers on an 18 GB drive (9 GB/container), each belonging to two different table spaces. Normally this setup would be sub-optimal, but if one of the table spaces is only accessed at certain times (for example, month-end), and the other is accessed during normal operation but not at month-end, then having containers share the same physical disk would not be a problem.

Apart from the Availability and Load considerations discussed, you may need multiple containers on older operating systems to get past a file size limit (such as 2 GB)

You also may need multiple containers per filesystem to reduce inode contention (AIX). One file for a heavy read/write table will result in inode contention; 2 or more containers on the same filesystem can help reduce this contention.

3.3.9.4 Considerations for separating data and indexes

Using DMS does not separate data and indexes into different table spaces unless you do so specifically to assign them to different buffer pools. If you need to ensure that certain indexes always remain in memory, then that would be a valid reason to move the indexes to their own table space and assign that table space its own buffer pool. Make sure you have sufficient disk I/O bandwidth to cope with this setup.

3.3.9.5 Table space names

With DB2 UDB v7.1 we introduce the ability to rename a table space — although this does not appear to improve performance as such, it can certainly save time! Currently, the name of a table space is specified by the user during table space creation, and there is no way of changing it during the lifetime of the table space object. If you wish to change the name of a table space, then the only way to accomplish this is through dropping and recreating objects. The objects within the table space and the table space itself need to be dropped.

The table space is then recreated using the new name, and the objects within the table space are then recreated. Obviously this is a big problem in terms of availability if the table space is very large or contains many data objects.

The new command will allow a user to rename the table space with a single SQL statement (for example, only updating catalog records and changing table space control information). An example of this would be:

```
RENAME TABLESPACE USERSPACE1 TO NEWSPACE1
```

Please refer to the *DB2 UDB SQL Reference,* SC09-2974 for more details.

Note

If you rename a table space, then you need to restore the table space from an old backup image taken before the rename command was run. The new name must still be used to perform the restore.

3.3.10 Other tips

- Try to keep the number of table spaces small, and keep multiple objects (with similar performance requirements) in each table space.
- Use DMS table spaces with raw device containers for user data and indexes as well as for the log files.

3.4 Buffer pools

It is no secret that accessing memory is faster than disk I/O. DB2 uses database buffer pools to attempt to minimize disk I/O. Database logging, however, is disk I/O which cannot be avoided, but which can be made more efficient; see 3.5, “Database logs” on page 78 for more details on this.

The most common question asked is, "How much memory do I need to allocate for buffer pools?" Although there is no definitive answer to this question, the general rule of thumb would be to start with about 75% of your system’s main memory devoted to buffer pool(s), but this rule has to be considered VERY carefully before being followed. The biggest factor for consideration is obviously the answer to the question, "Is this machine a dedicated database server?"

If the answer is no, then allocating 75% of system memory to DB2’s buffer pools is not going to be a good idea. It is the database administrator’s job to determine how much memory there is available on the system for DB2’s use, after allowances for the operating system and other applications have been made.

Many people want to know why the default buffer pool (IBMDEFAULTBP) is so small (1000 * 4 KB). The main reason is to allow you to connect to a database in situations where you may have over-allocated memory elsewhere. We do not recommend that you run a database using only the default buffer pool, but once connected to your database, you can then add additional buffer pools dedicated to specific table spaces.

When you add buffer pools to your database, the total memory required by all buffer pools must be available to the database manager to allow the buffer pools to be activated when the database is started. Should this memory not be available, then only the default buffer pool (`IBMDEFAULTBP`) is started along with one buffer pool for each table space which uses a page size different from the 4 KB default. This allows you to connect to your database. These extra buffer pools will be restricted to 16 pages. If this behavior was not the default, then you could potentially allocate too much memory to your buffer pools; and when you attempted to activate or connect to the database, the operation would fail.

Each buffer pool page allocated uses space in the memory allocated for the database heap (`DBHEAP`), for internal control structures. Therefore, if you increase your `BUFFPAGE` parameter and/or the total size of your buffer pools then also ensure that you remember to increase `DBHEAP`.

Note

If you mistakenly override the default buffer pool size and set it too big, resulting in a database start failure, then the registry variable `DB2_OVERRIDE_BPF` can be used to override the size of the default buffer pool. See the *DB2 UDB Administration Guide: Performance*, SC09-2945 for more details on this variable.

3.4.1 Mapping table spaces to buffer pools

You can create multiple buffer pools besides the default buffer pool `IBMDEFAULTBP`. You need to consider the following before determining the number of buffer pools your database should have.

3.4.1.1 Single buffer pool

In most situations, having one large buffer pool is the recommendation. Apart from its size, a single buffer pool needs no tuning. A single large buffer pool will also allow DB2 to efficiently utilize memory that has been allocated for buffer pool use.

If you decide to opt for table spaces using multiple page sizes, then you should create only one buffer pool for each page size. If you want to create table spaces using a page size other than the default 4 KB, then you will need a buffer pool which uses the same page size.

3.4.1.2 Multiple buffer pools

Multiple buffer pools, if badly configured, can have a huge negative impact on performance. When created for the right reasons, multiple buffer pools can improve performance; however, not all workloads will benefit.

You should consider defining multiple buffer pools when:

- You want to create tables which reside in table spaces using a page size other than the 4 KB default. This is required.
- You have tables which are accessed frequently and quickly by many short update transaction applications. Dedicated buffer pool(s) for these tables may improve response times.
- You have tables larger than main memory which are always fully scanned. These could have their own dedicated buffer pool. However, it needs to be large enough for prefetching requirements of such a large table (256 KB up to # MB). If these tables are scanned frequently then you could swamp the other buffer pools. (Tables this large will most likely always generate disk I/O irrespective of buffer pool size.)

Extended storage (see 3.4.3, “Extended storage” on page 76) was introduced in DB2 UDB v5.2 and is a buffer pool enhancement that can either help or hurt performance.

3.4.2 Buffer pool memory

There are two main ways to allow DB2 UDB to exploit AIX’s segmented memory address model. These involve setting some DB2 registry variables to free up two 256 MB segments of memory which will allow larger buffer pools to be created.

Set the registry variables `DB2_MMAP_READ` and `DB2_MMAP_WRITE` to `OFF` on AIX with SMS or DMS table space file containers. When you set these parameters off, the database manager:

- Frees up extra 256 MB segments
- Enables the use of the AIX JFS file system cache

An explanation of these variables is also given in the 3.4.2.1, “MMAP configuration parameters” on page 74.

Set the registry variable `DB2_FORCE_FCM_BP=NO`, which is the default for this variable. When it is set to `YES`, the Fast Communication Manager (FCM) buffers are allocated from the same shared memory segment. This gives optimal FCM performance when you use the Enterprise-Extended Edition of DB2 UDB and have multiple database partitions on one physical node. However, the Enterprise Edition of DB2 UDB will be working with only one database partition. By setting the variable to `NO`, we free another shared memory segment which can then be used for buffer pools.

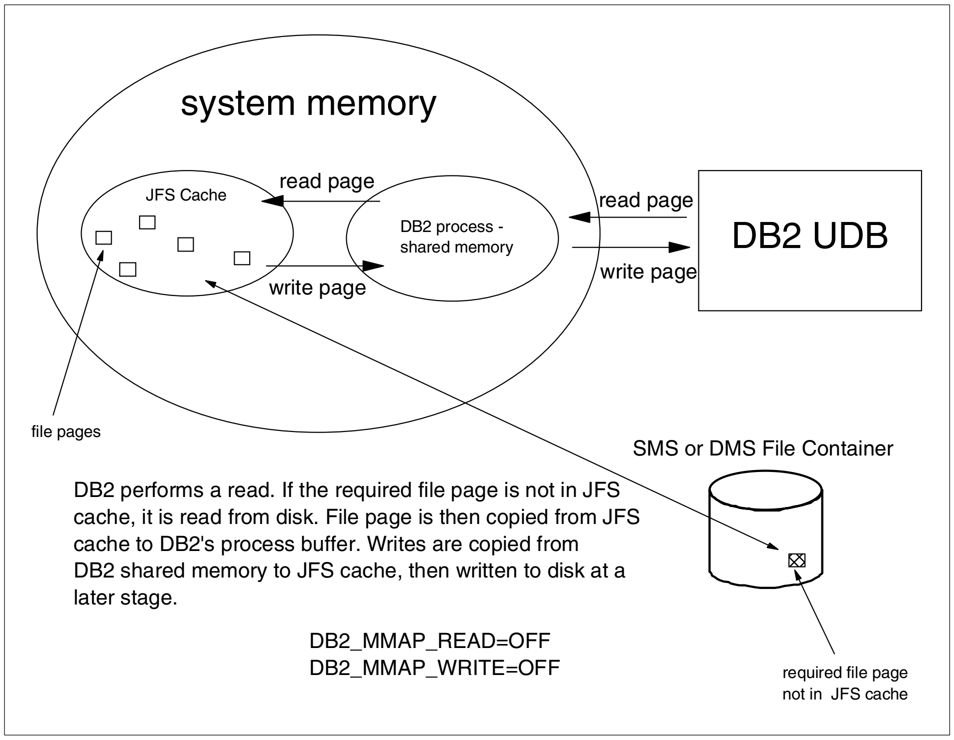
For customers using AIX 4.3.2 and later, the above points may not be as relevant. This is due to enhancements in shared memory segment size. Using AIX at this level and above will allow shared memory segments up to 2 GB in size.

Note

To eliminate database initialization overhead, which could, for example, be caused by activating large buffer pools, use the `ACTIVATE DATABASE <dbname>` command. This works well in a dynamic environment (such as a Web server) where all applications may frequently disconnect from the database.

3.4.2.1 MMAP configuration parameters

We have recommended setting these two registry variables to OFF. Here we will attempt to explain some of the reasons for this decision. The main advantage is that by setting these to OFF, we free up an extra memory segment which can be made available for the DB2 buffer pools. The following diagrams, Figure 18 and Figure 19, attempt to explain the differences made by setting these variables ON or OFF, and how this affects data page reads and writes for DB2 UDB.



DB2 performs a read. If the required file page is not in JFS cache, it is read from disk. File page is then copied from JFS cache to DB2's process buffer. Writes are copied from DB2 shared memory to JFS cache, then written to disk at a later stage.

Figure 18. Example with mmap variables set to OFF

In Figure 18 we see that when `DB2_MMAP_READ` and `DB2_MMAP_WRITE` are set to OFF, and we are using SMS or DMS file containers, the operating system's filesystem cache is utilized. Although at first glance, this situation seems like an obvious overhead, in reality this is not the case. By using the JFS (filesystem) cache we are letting the operating system take care of the I/O operations to disk.

Once the database performs a write, the changed file page is copied from the DB2 processes buffer in memory to the JFS cache. In most cases, data is not written immediately from the JFS cache, but the write is done at a later time. The filesystem log provides recoverability should the system crash while the filesystem cache still contains dirty pages.

The main advantage for DB2 is that this "late writing", which is known as *write-behind*, allows the database to continue processing without having to wait for an I/O request to complete.

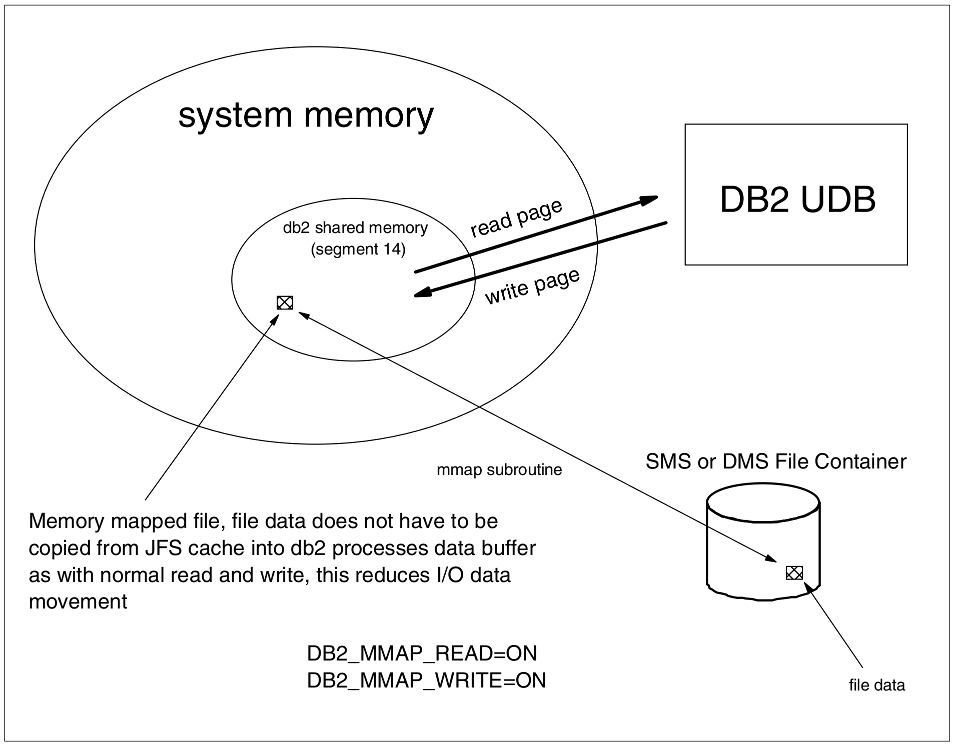

system memory
db2 shared memory
(segment 14)
read page
write page
DB2 UDB
mmap subroutine
SMS or DMS File Container
Memory mapped file, file data does not have to be copied from JFS cache into db2 processes data buffer as with normal read and write, this reduces I/O data movement
DB2_MMAP_READ=ON
DB2_MMAP_WRITE=ON
file data

Figure 19. Example with mmap variables set to ON

In Figure 19 we show the default behavior, which is to have both variables set to ON. Here we are utilizing mmap (memory mapped) I/O which allows files to be mapped directly to one of DB2's memory segments. When using mmap we bypass the filesystem and the usual operating system kernel buffers.

One of the main reasons mmap is implemented by default is that by mapping files explicitly and accessing it by address, the total cost of the system call overhead is reduced because the path length is shorter.

However, as we access the file by address, we do not use the read() or write() subroutines. In AIX the implication of this is that we do not benefit from the write-behind feature described earlier. The result is that modified pages may remain in memory and be written to disk in a random manner when the Virtual Memory Manager's (VMM) page stealer deems it necessary. If this happens, we could then see a number of small random I/Os which are inefficient.

Although this sounds like a major disadvantage, you must remember that if your database has been designed with performance in mind, the majority of table spaces should use DMS device containers. This means that the database manager is responsible for these operations and not the operating system. The performance impact in this case will be minimal.

3.4.3 Extended storage

Extended storage is essentially a secondary cache between the buffer pools and the disks (the buffer pools are the primary cache). On AIX this works by allocating extra shared segments (NUM_ESTORE_SEGS), each of which is the same size (ESTORE_SEG_SIZE).

Extended storage was introduced to exploit main memories larger than 4 GB on 32-bit systems. 4 GB was, on most platforms, the limit of virtual addressable memory for a 32-bit process. It is now common to see 32-bit systems with more than 4 GB of real addressable memory. The database manager cannot directly manipulate data that resides in the extended storage cache. However, it can transfer data from the extended storage cache to the buffer pool much faster than from disk storage.

Remember, though, that there is an overhead (CPU) when copying pages into extended storage (detach/attach costs and the cost of the copy). Every time a page is read out of extended storage, rather than disk, this saves the cost of the disk I/O (at the expense of CPU cost, as already mentioned). In I/O-bound systems, where workload is such that pages are frequently read from the extended storage, compared to number of writes, you will benefit. If you seem to be writing pages to extended storage, but never read them in (for example, if you are dirtying most pages) then there will be no performance gain, and you may actually impact your system's potential performance.

Extended storage can be enabled on a per-buffer-pool basis (using ALTER BUFFERPOOL) so if you have multiple buffer pools defined allocated to individual table spaces, then you can enable extended storage only for those table spaces which will benefit from the addition of extended storage.

3.4.3.1 When to use extended storage

You should use extended storage when your system/database/workload meet the following criteria:

- If your operating system's limit on the amount of addressable virtual memory for a process limits the size of the buffer pool, without utilizing all of the memory in the system. The larger the amount of memory available for to extended storage then larger the potential benefit will be (for example, when extended storage If your operating system's limit on the amount of addressable virtual memory for a process limits the size of the buffer pool, without utilizing all of the memory in the system. The larger the amount of memory available for to extended storage then larger the potential benefit will be (for example when extended storage becomes larger than the buffer pool).
- Workload is I/O-bound.
- Workload is read only/mostly — potentially only one copy goes to extended storage; subsequent page victimization re-uses the original copy. This works well if extended storage large enough to keep important tables/indexes fully cached (assuming these are too large for buffer pool). This scenario gives multiple reads for each copy in extended storage.

Therefore, if the workload is mainly read-mostly, or your workload involves large temporary tables which may otherwise spill to disk, then extended storage is a good choice.

3.4.3.2 When not to use extended storage

You should NOT use extended storage when your system/database/workload meet the following criteria:

- When the total number of pages allocated to extended storage (`num_estore_segs * estore_seg_size`) could alternatively be allocated to the buffer pool(s). The exception to this is, when running multiple buffer pools, you may want each buffer pool to have its own dedicated pages (`NPAGES`), but you want all buffer pools to share the extended storage area. We do not recommend this, however, because in most situations, defining a single buffer pool which is the same size as that of the dedicated buffer pools plus extended storage will outperform the other multiple buffer pool configuration. This occurs because by having one buffer pool, we do not incur the overhead of the extra copies involved when moving pages from buffer pool to extended storage.
- If you are CPU-bound, as described above, you are using up CPU cycles while trying to save disk I/Os. So if you are CPU-bound and not I/O-bound, extended storage will actually hurt performance.

- When the workload (or at least the portion in buffer pools that are configured to use extended storage) would not result in enough pages being read from extended storage, to offset the cost of placing the copies into extended storage.

3.4.3.3 Other extended storage considerations

The number of shared segments (NUM_ESTORE_SEGS) should be:

```
Memory for_estore / 256MB
```

Set extended storage segment size (ESTORE_SEG_SIZE) to:

```
Memory_for_estore / num_estore_segments_just_calculated
```

3.5 Database logs

In this section we will look at performance considerations relating to database logging. If set up inappropriately for your workload, database logging could become a significant overhead.

3.5.1 Why logging performance is important

Ignoring the performance of your database in relation to its logging can be a costly mistake, the main cost being time. The log files exist to provide the ability to be able to recover your environment to a consistent state and preserve the integrity of your data.

With the introduction of DB2 UDB v7.1, total log files can now be up to 32 GB in size, and you can perform extremely large amounts of work within a single transaction. With customers now using log spaces which are gigabytes in size instead of megabytes, the physical layout of the log files becomes very important. Placement of these files needs to be optimize, not only for write performance, but also for read performance, because the database manager will need to read the log files during database recovery.

Note

Prior to Version 7.1, the total log files must be up to 4 GB.

3.5.2 Filesystem or raw logical volume

We now have two options when deciding how to store the DB2's transaction log files, either in the operating system's filesystem or in a raw logical volume. The most familiar method here is to use the filesystem, as this is all we have supported in the past for logs. Using raw logical volumes for logging will bring performance improvements, similar to those gained from switching from table space file containers to DMS raw devices. However, it is worth pointing out that, as in most situations, there are advantages and disadvantages to using raw logical volumes for your log devices:

Advantages:

- The I/O path is shorter, since you are now bypassing the operating system's filesystem.
- Raw device striping may provide faster I/O throughput.
- No restrictions are imposed by a filesystem.
- A raw device can span multiple disks.

Disadvantages:

- The device (logical volume) created for the logs must be dedicated to DB2.
- The device (logical volume) created for the logs cannot be operated upon by any operating system utility or third-party tools which would back up or copy from the device.
- Currently, if using DPropR, the read log API will not call the user exit if you use the raw log device. The recommendation is that DPropR should not be used when using raw log devices. Similarly, do not use raw log devices if you use the `sqlurlog` API.

3.5.3 Mirroring

Always mirror your log devices; the log files are crucial to your database and you need to ensure that the disks they reside on are mirrored. It is recommended that the log disks be entirely dedicated to DB2 and that no other processes write to these disks. If possible, place the disks on separate disk controllers to maximize availability.

3.5.4 Placement on disks

Generally, we would recommend that the disks chosen for the log devices are the fastest disks in the system. Another consideration for performance is the placement of the container on the physical disk. To optimize performance for log devices, ensure that the logical device used by DB2 (filesystem or raw logical volume) is placed on the outer edge of the disk. There are more data blocks per track at this location on the disk. This location, therefore, lends itself to the typical activity of the DB2 logger process which writes to the log files, mostly with a large number of sequential reads and writes.

3.5.5 Number of log files

Your active log space will be determined by the number of log files you define. As we have seen already, the active log space is defined by the parameters (LOGPRIMARY + LOGSECOND) * LOGFILSIZ. You need to take into account your database workload and therefore the potential logging activity.

If, for example, you have an OLTP type workload, then you would not obtain good performance from log files that are too small, as you would fill them very quickly. In this case, if a user exit were enabled, this would mean more work for the database manager.

Most DSS workloads with little update, insert, or delete activity may benefit from a smaller number of log files. However, in this environment, where logging activity is low, it may be worthwhile using a larger log file size to reduce administration overhead.

3.5.6 Size of logs

Remember that the total active log space can now be up to, but not including, 32 GB. So the calculation would be:

```
( LOGPRIMARY + LOGSECONDARY ) * LOGFILSZ * 4k < 32GB
```

In this formula, LOGFILSIZ represents the Log File Size database configuration parameter.

If you are using raw devices for logging, then remember that log records are still grouped into log extents; the size of each is LOGFILSIZ (4 KB pages). DB2 places the log extents consecutively in the log device, and every log extent carries with it a 2-page overhead for extent header information. This affects the number of log extents which can be written to your raw log device. The total number can be calculated by the formula:

```
raw-device-size / (logfilsz + 2)
```

The device must be large enough to support the active log space that you are going to allocate. By this we mean that the number of log extents which can be contained on your raw device must be greater than (or equal to) the value of LOGPRIMARY.

For more details on this, please refer to the *DB2 UDB Administration Guide: Implementation*, SC09-2946.

3.5.7 Flushing logs during on-line backup

Since Version 7.1, the active log is closed when the on-line backup completes. This can significantly reduce log management during on-line backups. You do not have to do anything to use this new feature, as it is the default behavior. To disable it, use the `DB2_DISABLE_FLUSH_LOG` registry variable; set this to `ON`. You may want to disable this feature if:

- You have limited disk space. Each time an active log file is truncated, an amount of space is left unused with the physical log file. If you perform a very large number of on-line backups each day, you may wish to disable the inclusion of the last active log file. The amount of space wasted will obviously depend on the size of your log files.

  For example, if your log files are 50 MB each and you take 4 on-line backups per week, and on average, each time the active log was truncated you wasted 50% of the space, then you would waste a total of 5.2 GB over one year. Note that truncating 50% each time is just an example, and the actual amount depends on how much of the current active log file has been used.

  You will need to trade off the advantage of having an on-line backup and all the required logs against the amount of space wasted.

- You may also wish to disable the inclusion of the last active log file if you find you are receiving Log Full messages a short time after the completion of the on-line backup. When a log file is truncated, the reserved active log space is incremented by the amount proportional to the size of the truncated log. The active log space is freed once the truncated log file is reclaimed. The reclamation occurs a short time after the log file becomes inactive. It is during the short interval in between that you may receive Log Full messages.

  For example, assume you have 100 MB of active log space (`LOGPRIMARY` + `LOGSECOND`) * `LOGFILSIZ`. Then all logging required by concurrent, active units of work (UOWs) must fit in this space.

  As an example, assume that we have a unit of work ‘A’ which starts and initially writes 40 MB to active log space. Just after this UOW ‘A’ starts, an on-line backup of this database completes, and by default, will now truncate the current active log file. Assume that, as a result of the truncation, we have just wasted 10 MB (this is what space was remaining in the physical log file); this 10 MB a moment ago was part of active log space.

But our UOW ‘A’ has still not completed, so the 10 MB of truncated space reduces available active log space by 10 MB. This means at this point in time, active log space would actually be 50 MB (40 MB which UOW ‘A’ wrote + 10 MB truncated space) and not 60 MB.

So if another UOW ’B’ tried to start at this point, and required MB of log space it will receive SQL0964. The reason here is that the first UOW ‘A’ has not completed. Once the UOW ‘A’, which commenced before the log truncation took place, has completed, space reclamation will begin.

Note

SQL0964 ’Log Full’ is a logical error which indicates that active log space is full. This is different from the physical log file being full.

3.5.7.1 Pre-version 7 behavior

While performing on-line backup, all activities taking place in the database are logged.

When an on-line backup is restored, the logs must be rolled forward at least to the point in time at which the backup was completed.

The log file used at backup time may continue to be open long past the time of the backup completion.

3.5.7.2 Version 7 behavior

Changes are made in the on-line backup such that the active log is closed when an on-line backup completes.

If `USEREXIT` is `ON`, the active log will be archived.

3.6 Before creating a database

Once the physical space allocation is complete in the form of logical volumes and you have decided what table spaces and tables will be created, the next step is to actually create the database and table spaces.

We suggest making the create database statement simple, using as many defaults as practical. Most resources, such as table spaces and containers, can be added to the database once the basic database structure exists.

3.6.1 Number of instances

The number of instances you define, and therefore databases, will only play a major part in performance tuning if you over-allocate system resources by having too many large databases on the same machine. Effective capacity planning needs to be performed before adding production instances and databases to a server which may already contain another DB2 production environment.

Whenever adding a new production database, take into account future requirements, for example:

- Will you need additional CPUs, memory, disks, or adapters for your current system?
- Can you afford the maintenance window to install these if required?
- Should you consider running the new database on its own dedicated server?

3.6.2 Number of databases per instance

In a production environment, we would generally recommend that you create one database per instance. This brings a number of advantages, for example, there is isolation from errors between databases, database manager configuration parameter changes affect only one database, information in the `db2diag.log` file is dedicated to the database, and configuration changes can be made without affecting other instances or databases.

Chapter 4. Database design

So far, we have discussed the physical layout of the disk storage, the table space design, and the layout of buffer pools in the previous chapter. The next major consideration, in order to obtain acceptable response times, is the design of tables and indexes. When you design tables, you must choose an appropriate data model and data types for them. With respect to indexes, you need to be aware of the advantages and disadvantages of creating indexes, and create appropriate ones.

DB2 UDB allows the database administrator to choose from a variety of options when creating tables and indexes, and it is essential for the database administrator to understand the various advantages and disadvantages of choosing a particular option.

In this chapter, we will explain how each of these options affects database performance, and provide helpful information that a database administrator can use to design new tables and indexes or improve existing tables and indexes.

4.1 Tables and performance

Once the table spaces have been created and configured, we can begin to look at what design factors will influence the performance of the tables themselves. In this section we will look at these factors.

4.1.1 What to consider before creating tables

Before creating tables, you should draw a logical design of your database. In this section we discuss aspects of logical database design that affect performance.

4.1.1.1 Normalizing

Normalization reduces redundancy from your data and can improve the performance of update and delete statements, since you only have to do it in one place. By normalizing your data, you try to ensure that all columns in the table depend on the primary key.

There are four “forms” of normalization (first, second, third, and fourth). We will show an example of the third and fourth forms; however, examples of all four forms can be found in the *DB2 UDB Administration Guide: Planning*, SC09-2946.

First normal form

In first normal form, each cell in the table only contains one value.

Second normal form

In second normal form, each non-key column is dependent upon the *entire* key and not just part of the composite key.

Third normal form

In third normal form, each non-key column is dependent upon the key column as in the second form, but is also independent of other non-key columns.

In Table 3 we can see that the `DeptName` column is dependent on the `DeptNo` column, so this table does not conform to the third normal form. If we update `DeptName` for one employee, this would not update the `DeptName` column for other employees belonging to the same department.

Table 3. Table not in third normal form

EMPNO (Primary Key)	**LastName**	**DeptNo**	**DeptName**
001	Brown	01	Support
002	Smith	02	Sales

In this basic example using normalization to conform to the third normal form, would produce two tables (Table 4 and Table 5). Now updating the `DeptName` column is easy, as we just update column in Table 5.

Table 4. Table in third normal form (1)

EMPNO (Primary-Key)	**LastName**	**DeptNo**
001	Brown	01
002	Smith	02

Table 5. Table in third normal form (2)

DeptNo (Primary-Key)	**DeptName**
01	Support
02	Sales

Fourth normal form

In fourth normal form, no row in the table must contain two or more multi-valued facts about an entity. In Table 6 we are displaying two relationships, one between `EMPNO` and `Project`, and one between `EMPNO` and `DevLang` (programming language used). Since this table represents both relationships, it does not conform to the fourth normal form.

Table 6. Table not in fourth normal form

EMPNO (Primary-Key)	Project (Primary-Key)	DevLang (Primary-Key)
001	Budget Tracker	C++
001	Customer Database	C++
001	Budget Tracker	Java
001	Customer Database	Java

To fix this problem, we need to represent this information in two separate tables, Table 7 and Table 8.

Table 7. Table in fourth normal form (1)

EMPNO (Primary-Key)	Project (Primary-Key)
001	Budget Tracker
001	Customer Database

Table 8. Table in fourth normal form (2)

EMPNO (Primary-Key)	DevLang (Primary-Key)
001	C++
001	Java

In this example, it is important to remember that if only certain programming languages (`DevLang`) were used for certain projects, then the values in these columns would be interdependent, so the original table, Table 6, should not be split into the two separate tables we created. One consideration when normalizing tables is that the more tables you create, then the more likely it is that the JOIN requirements between these tables will impact query performance.

Note

It is our recommendation to start with tables in the fourth normal form, then measure the performance. If performance is unacceptable, structure the table in the third normal form, then reassess performance.

4.1.1.2 Denormalizing

With normalization, we are striving to achieve a database implementation where each item in the database is contained in one place and is not duplicated. Traditionally this has been done to improve update and delete performance. The disadvantage is obviously data retrieval, specifically when a query is accessing a large number of related pieces of data from different tables.

Denormalization is the process of placing the same piece of information in more than one place in an effort to improve retrieval performance. If you want to denormalize your tables in an attempt to improve performance, consider the following:

- Can the database be tuned in some other way to improve performance without denormalizing?
- What are the likely performance gains by denormalizing, and will these be a reasonable trade-off against the added update overhead?
- Can we easily revert back to normalized tables if the performance gains are not what we expected?

Note

In DSS environments where the data is never updated once it is loaded, denormalizing can be a good choice, since it speeds up queries.

4.1.1.3 Data types

Data types are a column attribute definition used when a table is created. The data type tells us what kind of data will be held in the column and what the length of the values in it will be.

In DB2 UDB, data types can be categorized as:

- DB2-Supplied Data Types: These can be Numeric, String (Binary, Single Byte, Double Byte), or DateTime.
- User Defined Data Types: These can be User Defined distinct Types (UDTs), User Defined Structured Types, or User Defined Reference Types.

For more information on data types, please refer to the *DB2 UDB Administration Guide: Implementation*, SC09-2944, and the *SQL Reference*, SC09-2974.

From a performance perspective, the use of User Defined Data Types will not impact response times. For example, User Defined distinct Types share the same code that is used by built-in data types, to access built in functions, indexes, and other database objects.

4.1.1.4 Use VARCHAR or LONG VARCHAR

In the past, the data type `LONG VARCHAR` has been used to support columns of up to 32,700 bytes. However, the limit for `VARCHAR` columns is now 32,672, which should be taken into account when designing tables, as it is now worth considering using `VARCHAR` instead of `LONG VARCHAR`. For example, when an expression returns a varying length string using `LONG VARCHAR`, these expressions cannot be used in:

- Basic quantified `BETWEEN` or `IN` predicates
- Column functions
- `VARGRAPHIC`, `TRANSLATE` and datetime scalar functions
- Pattern operand in a `LIKE` predicate
- String representation of a datetime value

As a result of these restrictions, in most cases it makes more sense to use the `VARCHAR` data type and impose fewer restrictions on the expression.

- One final and very important consideration is that data stored in `LONG VARCHAR` columns is **not** buffered in the database buffer pool. The `VARCHAR` data pages, however, are stored in the buffer pools.

Note

As of DB2 UDB V7.1, the data types LONG VARCHAR and LONG VARGRAPHIC (not discussed here) will be marked as deprecated. This data type will be supported, but enhancements may not be implemented for this data type. If you are defining items such as structured types, do not define them using LONG VARCHAR or LONG VARGRAPHIC types. Use the CLOB or DBCLOB data type instead.

4.1.1.5 Why define columns as NOT NULL?

The term NULL signifies an unknown state. Calculations performed on columns containing null values will result in an unknown outcome. Nearly all data types in DB2 support null values.

A column can reject null values when you specify the NOT NULL clause in the column definition of the CREATE TABLE statement, for example:

```
CREATE TABLE TABNULL (col1 DECIMAL(1,2) NOT NULL WITH DEFAULT 10)
```

In the example shown the column col1 will not accept null values, and a known value must be entered. You can also specify a default value to be inserted by combining the WITH DEFAULT option and NOT NULL options of CREATE TABLE. When this is done, if a row is inserted and a value is missed for the column using the NOT NULL WITH DEFAULT definition, then the value used for the WITH DEFAULT clause will be inserted, as we have specified that null values will not be accepted.

In general, columns defined as NOT NULL perform better than nullable columns. The reason for the performance gain is the path length reduction; the database engine does not have to check for null values in a NOT NULL column. It is worth mentioning that there are space savings as well when using NOT NULL. Every nullable column requires one extra byte per column value. By defining columns as NOT NULL, there will be space savings which may lead to a reduced number of used tables, index pages, and index levels, which can improve query performance.

4.1.1.6 Identity columns

Significant performance enhancement can be realized using DB2 generated identity values compared to those implemented by an application.

Identity columns are ideally suited to the task of generating unique primary key values. Applications can use identity columns to avoid the concurrency and performance problems that can result when an application generates its own unique counter outside the database. For example, one common application-level implementation is to maintain a 1-row table containing the counter, and having each transaction lock this table, increment the number, and then commit (unlock); that is, only one transaction at a time can increment the counter. In contrast, if the counter is maintained via an identity column, then much higher levels of concurrency are achievable because the counter is maintained by DB2 and is not locked by transactions, and thus one uncommitted transaction that has incremented the counter will not prevent other subsequent transactions from also incrementing the counter.

The counter for the identity column is incremented (or decremented) independently of the transaction. If a given transaction increments an identity counter two times, that transaction may see a gap in the two numbers that are generated because there may be other transactions concurrently incrementing the same identity counter (inserting rows into the same table).

If an application must have a consecutive range of numbers, that application should take an exclusive lock on the table that has the identity column. This decision must be weighed against the loss of concurrency to that table from other applications. Furthermore, it is possible that a given identity column can appear to have generated gaps in the number, because a transaction that may have generated a value for the identity column had rolled back, or that the database which has cached a range of values has been deactivated (normally or abnormally) before all the cached values were assigned.

The sequential numbers that are generated by the identity column have the following additional properties:

- The values can be of any exact numeric data type with a scale of zero. (SMALLINT, INTEGER, BIGINT, or DECIMAL with a scale of zero). By contrast, single and double precision floating point are considered approximate numeric data types and are not allowed as identity columns.
- Consecutive values can differ by any specified integer increment. The default increment is 1.
- The counter value for the identity column is recoverable. If DB2 should fail, the counter value is reconstructed from the logs, thereby guaranteeing that unique values continue to be generated across a DB2 failure.
- Identity column values can be cached to give better performance.

Each table may have a single column that is defined with the IDENTITY attribute. Some examples of using an identity column would be an order number, employee number, stock number, or incident number. The values for an identity column can be generated by DB2 using the clauses ALWAYS or BY DEFAULT. Following is an example of this; see *DB2 UDB SQL Reference,* SC09-2974 for more details:

- `CREATE TABLE TAB1 (col1 INT, col2 INT, coluniq INT GENERATED ALWAYS AS IDENTITY (START WITH 0, INCREMENT BY 1))`

Note

At the time of writing, Identity Columns are not supported in an EEE database with more than one partition. Please check the Release Notes for availability of this feature for EEE databases.

4.1.1.7 Generated columns

A generated column derives the values for each row from an expression, rather than from an insert or update operation. Although an update or insert trigger will produce a similar effect, the advantage of using a generated column is that the value(s) derived are consistent with the expression used in the CREATE TABLE or ALTER TABLE statements.

You can use generated columns to improve the performance of queries in which the evaluation of a expression must be done many times, or if the computation of the expression is complex. Query performance can be improved more by creating indexes on the generated columns.

The GENERATED ALWAYS AS clause is used to create a generated column in a table, and you include with this the expression to be used to create the column. The GENERATED ALWAYS AS clause can be used with either the CREATE TABLE or ALTER TABLE statements.

The following example shows an example which creates a table with two regular columns COL1 and COL2 and a generated column COL3 which is derived from the two regular columns.

```
CREATE TABLE TABLE1
(COL1 INT, COL2 INT, COL3 INT
GENERATED ALWAYS AS (COL1 - COL2))
```

Suppose that you now populate the table and wish to use the ALTER TABLE statement to add a fourth column which is also a generated column. You must first alter the table's state by turning integrity checking OFF.

```
SET INTEGRITY FOR TABLE1 OFF
```

This needs to be done for TABLE1, as the table includes generated columns which generate values based on other columns in the table. We can now add another generated column to this table.

```
ALTER TABLE TABLE1
ADD COLUMN COL4 INT GENERATED ALWAYS AS
(CASE WHEN COL1 > COL2 THEN COL3 ELSE NULL END)
```

Now the new column has been added, we need to turn integrity checking back ON.

```
SET INTEGRITY FOR TABLE1 IMMEDIATE CHECKED FORCE GENERATED
```

The FORCE GENERATED clause is used because TABLE1 includes generated columns, and the values for the new column will be computed based on the expression stored in the column.

For more detailed information on the SET INTEGRITY command, see the *DB2 UDB SQL Reference*, SC09-2974.

Generated columns and indexes

Indexes can also be defined on generated columns. As an example, suppose you have created a simple table and index using the following syntax:

```
CREATE TABLE TAB1 (NAME CHAR(20))
CREATE TABLE INX1 on TAB1(NAME)
```

Now let us assume that values were inserted in to the `NAME` column using both uppercase and lowercase letter, so that when we ran the statement `SELECT * FROM TAB1`, we saw the output as follows:

```
SELECT * FROM TAB1

NAME
-------------------
JON smith
mark JONES
SARAH THOMPSON
Simon Woods

4 record(s) selected.
```

As you can see above, the values in the `NAME` column contain both uppercase and lowercase lettering. Although this scenario might be unlikely in most business applications, let us assume that you now want to perform another select on the table but want the search to be case insensitive. In this case, your application would run a statement such as `SELECT * FROM TAB1 WHERE LCASE(NAME)='mark jones'`. This query has a predicate with a specific value, but the database manager cannot determine the start/stop index key value to scan on the index. This is because each of all rows needs to be translated into lower case by `LCASE` function, and then evaluated as to whether it is `'mark jones'`. This is not a problem if the `TAB1` table is small; however, if it contains thousands of rows, this processing cost would be considerably greater.

One solution would be to created a generated column which converted the values stored in the `NAME` column to either uppercase or lowercase. Once this was done, you could create a unique index on the generated column, as we are allowed to define indexes on generated columns. To do this for the table in our example, we would do the following, using the same principles regarding `SET INTEGRITY` that we discussed earlier in this section:

```
SET INTEGRITY FOR TAB1 OFF;
ALTER TABLE TAB1 ADD COLUMN NAMECASE CHAR(20) GENERATED ALWAYS AS
(LCASE(NAME));
SET INTEGRITY FOR TAB1 IMMEDIATE CHECKED FORCE GENERATED;
CREATE UNIQUE INDEX IND1 ON TAB1 (NAMECASE);
```

In this example, we first put the table in to the correct state to allow the generated column to be added, we then add the column, turn integrity checking back on and force the column to be populated with values using the criteria in the ALTER TABLE statement. Finally, a unique index is created on the generated column. This new index can then be utilized when the query is issued which needs to perform case insensitive searches on names. The SELECT * FROM TAB1 would now return the result set as follows:

```
SELECT * FROM TAB1

NAME                NAMECASE
------------------- --------------------
JON smith           jon smith
mark JONES          mark jones
SARAH THOMPSON      sarah thompson
Simon Woods         simon woods

4 record(s) selected.
```

Now you can re-write the query SELECT * FROM TAB1 WHERE LCASE(NAME)='mark jones' to SELECT * FROM TAB1 WHERE NAMECASE='mark jones' so that the index IND1 on the NAMECASE column can be used.

This example is used to simply demonstrate one of the ways in which using a generated column can improve the performance of your applications or queries. Although the generated column formula used was simple, this example may provide ideas for how this column type could be used in your environment.

4.1.1.8 Large objects, LONG data, and buffer pools

Where appropriate, consider using a larger page size and VARCHAR or VARGRAPHIC data type instead of large objects (CLOB, DBCLOB) or LONG data (LONG VARCHAR, LONG VARGRAPHIC). As has been stated previously in this chapter, large objects and LONG data are not buffered in DB2's buffer pool(s), so you will have to rely on the operating system's filesystem cache for buffering.

4.1.2 LOB considerations

Large objects (LOBs) are mainly used to store data types such as audio, video, and large drawings. This type of data will generally be too large to store using the LONG VARCHAR or LONG VARGRAPHIC data types, which each have a 32K limit. DB2 supports three different LOB data types:

- Binary Large Object (BLOB)
  Used to store binary strings with no associated code page.

- Character Large Object (CLOB)
  Mostly used to store single byte character strings (mostly text) which are, or could become too large for a VARCHAR.
- Double-Byte Character Large Object (DBCLOB)
  Used to store a character string made up of double-byte characters; the main type of data stored will be text information.

The LOB data types summarized above can be used to store data objects as strings up to 2 GB in size. A single table row can contain up to 24 GB of LOB data, and the table itself may hold a total of 4 TB of LOB data.

From a performance point of view, if you anticipate that you will be storing a large amount of LOB data in your tables, then you may want to ensure that you use DMS table spaces to hold these tables (SMS can also be used). The advantage of using DMS is that you are able to specify an alternative table space, using the LONG IN clause of CREATE TABLE. If using DMS, then use file container types to hold your LOB data. Remember that LOB data is NOT stored in the buffer pool, so by using the DMS file containers (or SMS table space) you will benefit from the operating systems file system cache.

Note

You can only specify an alternative table space to hold when the table is first created

4.1.2.1 Logged or not logged

When using the CREATE TABLE statement, you define column attributes for that table. If you are creating columns to be used for LOB data, then there is an option to allow you to decide whether or not you want to log any changes made to that LOB column. If you choose the LOGGED option, then any changes made to the column will be logged by DB2, this means that changes are recoverable. However, we do not advise using the LOGGED option for LOBs greater than 10 MB (note that LOBs greater than 1 GB cannot be logged). During database recovery, any “not logged” LOB data will be replaced with binary zeros.

Note

To create strings greater than 1 GB using the BLOB, CLOB, or DBCLOB data types, you must use the NOT LOGGED option in your column definitions.

4.1.2.2 Compact or not compact

Other LOB options available which will affect performance are the COMPACT or NOT COMPACT clauses of CREATE and ALTER TABLE. If you define your LOB column as COMPACT, then any values inserted into the LOB column will occupy minimal disk space. To do this, the LOB data object is split into smaller segments. Normally a LOB data objects is a 64 MB area split up into segments. Each segment will be 1024 bytes in size or any power-of-two multiple of this value (2048,4096 up to 64 MB). By using COMPACT, we can allocate smaller segments than these and allocate the minimum required space (space used is rounded to the nearest 1, KB). By allocating smaller segments, we now have very limited free space in the LOB data object which could be used for any future append operations. This could lead to a performance degradation when performing this type of operation.

If you choose the NOT COMPACT option, then when you insert LOB data, some free space is also inserted to assist in any future appends to this LOB data. The NOT COMPACT option is the default option, and as most data defined as LOB will be large in size, we would recommend using this default to take advantage of the append performance improvement.

4.1.3 Creating tables

When creating tables, there are a number of decisions which need to be made that will influence the performance of each table.

4.1.3.1 Which table spaces?

There are two main performance considerations when assigning tables to table spaces: one is expected activity, and the other is expected size. Obviously, these considerations will also be affected by any capacity planning that you have done.

Expected activity is basically trying to understand how your table will be used, if it will be heavily accessed and updated, if it will be read mostly, or if it will be a table for historical/reference data that may be accessed only at certain times of the year.

Try to categorize your tables and group them into table spaces accordingly. For example, small infrequently accessed tables could be grouped together in a single table space with a small buffer pool (or you could let them use the default buffer pool). Larger tables which you expect to be heavily accessed may be better in their own dedicated table space with its own buffer pool.

However, we are not recommending grouping all large tables in one table space, as this would in most cases degrade performance. One example would be a mixed workload where you have two large base tables, one which is updated only once per month but is used heavily for read-only queries, and another large table which is updated frequently each day. These tables will perform better in their own table spaces, as you will be able to configure each table space for each type of table workload (DSS and OLTP) by setting an appropriate extent size, prefetch size and using a suitable page size in each case.

4.1.3.2 Why plan free space when creating tables?

Planning to maintain free space in your tables can affect the performance of your database. That is to say, by not planning free space, you may see a performance drop when inserting data into the table.

When you insert data, a default `INSERT` search algorithm is used by DB2 to perform the inserts. The algorithm uses Free Space Control Records (FSCRs) to find pages with enough space. Even if the FSCR finds a page which has enough free space, the space may not be usable if it is "reserved" by an uncommitted `DELETE` from another transaction. This is one of the main reasons we recommend that applications `COMMIT` frequently to free this space.

Not all of the FSCRs in the table are searched when an insert is performed. The `DB2MAXFSCRSEARCH=N` registry variable limits the number of FSCRs visited for an insert (default=5).

Note

You need to modify the value of `DB2MAXFSCRSEARCH` to balance insert speed with space reuse. Large values optimize for space reuse. Small values optimize for insert speed. Setting the value to -1 forces the database manager to search all free space control records.

After searching five FSCRs, if we have not found space, the INSERT is simply appended to the end of the table. Once this happens, subsequent inserts also append to the table until two extents are filled, then the search process repeats. Each search then starts at the FSCR where the last search ended.

Once the entire table has been searched, we append without searching until space is created elsewhere in the table, via a DELETE, for example.

To ensure that you have free space on the data pages in your tables, use `ALTER TABLE PCTFREE <value>` before a `LOAD` or `REORG` operation. This will ensure that the percentage value specified for `PCTFREE` will be left on each page during the LOAD or REORG. Free space is important if you are going to consider using clustered indexes, which are discussed later in this section.

Other search algorithm options are APPEND MODE and clustered indexes, which are discussed in 4.1.3.3, “Why use append mode?” on page 98 and 4.2.4, “Clustering indexes” on page 109.

Another consideration for having free space is overflow records. These are created by updates that enlarge existing rows such that the row no longer fits on its page; the record is then inserted on another page (with enough free space). This is an overflow record. The original RID (Record Identifier) is then converted to a pointer record, which contains the overflow record's RID. However, indexes keep the original RID, which means that for overflow records, an extra page read is required to access data — this degrades performance.

Table reorganization (REORG) will eliminate overflow records.

4.1.3.3 Why use append mode?

Append mode is basically a search algorithm option used when inserting data. This mode can be switched ON using the ALTER TABLE statement with APPEND MODE clause; the default is OFF. If set to ON, then you must not have a clustered index defined on the table, as data will be appended to the end of the table and no free space information will be maintained. If append mode is set back to OFF, make sure you reorganize the table to update the free space information in the Free Space Control Records (FSCRs).

Since no free space information is maintained when append mode is used, if records are deleted from the table whose append mode is on, this free space will not be reused. Thus, you should use append mode for the tables which only grow (for example, journals, history tables, and so on).

Where insert performance is a key criteria, then define your table with APPEND MODE set to ON. New rows will be inserted at the end of the last extent, and no searching for available space will be performed; neither will updates to FSCR records.

There is one important consideration when using APPEND MODE set to ON. Although inserts will be faster, more space will be needed (in most cases) compared to using APPEND MODE set to OFF or by using a clustered index to try to insert records on the same page as other records with the same index key value.

Note

If APPEND MODE is ON, then the PCTFREE value for the table cannot be greater than zero.

4.1.3.4 What about “in-place” reorg?

By running the REORG utility, we reorganize table data into a sequence defined by one of your table’s indexes in an attempt to improve performance. Because the REORG utility uses temporary tables which can be larger than the base table being reorganized, temporary table spaces are normally used to perform the reorganization. Another alternative, which will improve the performance of table reorganization, is to try leave free space in your table space.

If you leave enough free space in each table space to REORG the largest table, then you will be able to perform the reorg "in-place" (in the same table space). A performance benefit will be gained, because DB2 will not have to copy the data to the temporary reorg files in the other table space and then copy them back again to the base table. Remember, though, that the temporary tables will still be larger than the base table, even if created in the same table space.

In summary, for optimal performance, do the reorg "in-place" if you can spare enough free space in the target table space.

4.1.3.5 Using NOT LOGGED INITIALLY

The NOT LOGGED INITIALLY option of the CREATE TABLE and ALTER TABLE statements allows you to turn off logging for a table for that unit of work. A table created using this option is commonly known as a “not-logged” table. The table is still classed as a regular table and has an entry in the catalog tables. When data is changed in the same unit of work as the CREATE TABLE statement, in a “not-logged” table, then these changes are not logged. Once a COMMIT statement is issued, the “not-logged” state is deactivated.

The “not-logged” state can be reactivated, by issuing the ALTER TABLE statement with ACTIVATE NOT LOGGED INITIALLY option. Changes occurring in the same unit of work as the ALTER TABLE statement are not logged.

Note

You can only use the ALTER TABLE statement to activate the “not-logged” state if the table was originally created with the NOT LOGGED INITIALLY option.

Because the “not-logged” state is only active in the unit of work that was generated by the CREATE TABLE or ALTER TABLE statement, only the application that issued the CREATE TABLE or ALTER TABLE statement can access and change the table without logging. During this unit of work, a Z table lock is held on the table.

See the *DB2 UDB Administration Guide: Implementation*, SC09-2944 for more information about the lock types of DB2 UDB.

Remember, if “not-logged” status is activated, any changes performed in this unit of work cannot be rolled back. Moreover, you cannot recover “not-logged” tables using roll-forward recovery. Therefore, you should activate “not-logged” status when you can re-create the table, even though it is damaged, for example, when you are:

- Creating a large table, using data from a source such as another table or file, and you do not want the population of the new table logged.
- Creating a summary table using the REFRESH DEFERRED option and you do not want the update by REFRESH TABLE statements to be logged.

To re-enable roll-forward recoverability of the table, it will have to be backed up again as part of either a table space or database backup.

Note that the NOT LOGGED INITIALLY option does not imply better performance. Although you can reduce logging overhead using this option, the data pages changed must be flushed to disk at the commit time because changes are not logged, and the commit can take a long time. If the buffer pool assigned to the table space to which the table belongs is small, most of the changed pages would have to be written to disk, and using the NOT LOGGED INITIALLY option may improve performance. However, if the buffer pool is large enough for the changed page not to be written to disk, using the NOT LOGGED INITIALLY option may decrease performance. The NOT LOGGED INITIALLY option is intended to overcome log space limitations. You may want to activate NOT LOGGED INITIALLY if you want to perform large changes where normal logging would give the log-full condition (SQL0964).

4.1.3.6 Declared temporary tables

Before DB2 UDB V7.1 (on UNIX, Windows, and OS/2), we supported two basic physical tables. Most common was the “regular” or persistent table. These tables exist in the database as entries in the system catalogs and reside in a user defined table space. The other type of table is commonly known as system temporary tables. These table types are created in a temporary table space by the database manager when processes or the optimizer require it. Use of these tables is restricted to the application that generated the table and no entries exist in the system catalogs for these tables. Temporary tables also only have a life span that lasts until the end of the statement that generated them.

With DB2 UDB V7.1 we introduce a new type of temporary table known as a declared temporary table. A declared temporary table is a temporary table which is available externally to the users via SQL. However, from the application perspective, only the application that creates the table may access the table, and the table will exist as long as the application is connected to the database.

These types of tables are very useful when you are running a complex SQL query which needs to process large amounts of data. By creating intermediate tables that are used to process the data and are then made available for the duration of the applications connection, you reduce the need to rebuild these intermediate tables; this results in a performance improvement.

You can create a temporary table using the `DECLARE GLOBAL TEMPORARY TABLE` statement. At this point the table will exist, but will contain no rows. This kind of temporary table is called a declared temporary table.

A declared temporary table persists within the current application’s connection and is dropped implicitly at the termination of the connection. During the connection period, the application can select rows from the declared temporary table, perform `INSERT/UPDATE/DELETE` statements without logging (as no logging is performed on this table type), and even drop the table explicitly. You can use declared temporary tables from the standard SQL interfaces such as ODBC, CLI, and static/dynamic SQL.

Defining a declared temporary table

Here is an example of creating a declared temporary table (assuming the application is connected to a database):

```
DECLARE GLOBAL TEMPORARY TABLE DTT1 (employee CHAR(12),salary int)
   ON COMMIT RESERVE ROWS NOT LOGGED;
```

This example creates a declared temporary table DTT1. The schema SESSION is assigned when a declared temporary table is created. The ON COMMIT RESERVE ROWS clause is used to allow the rows in the declared temporary table to persist after a commit. The other clause you can specify is the ON COMMIT DELETE ROWS clause (it is the default value). You are also required to specify the NOT LOGGED clause. Note that other applications can create their own declared temporary table called SESSION.DTT1 at the same time. Although these are different tables, they can use the same table name because a temporary table does not have any entries in the system catalog tables.

Note

Before defining a declared temporary table space, you must create a user temporary table space in advance using CREATE USER TEMPORARY TABLESPACE statement. This is a new table space type introduced with DB2 UDB V7.1 to allow greater control over the physical placement of this type of table.

The following shows a simple example of how to manipulate the declared temporary table we created.

```
INSERT INTO SESSION.DTT1
 SELECT employee,salary FROM employee WHERE salary>15000;

SELECT count(*) FROM SESSION.DTT1;

COMMIT;

SELECT count(*) FROM SESSION.DTT1;

CONNECT RESET;

CONNECT TO sample;

SELECT count(*) FROM SESSION.DTT1;
```

First, a subset of the employee table is retrieved and inserted into the declared temporary table DTT1. Notice that the schema name of the declared temporary table is SESSION and the inserted rows are **not** logged. Then two SELECT statements are performed for DTT1 before and after a COMMIT statement. Both of them get the **same** result. The third SELECT statement after reconnecting to the sample database would get an error (SQL0204), as the declared temporary table DTT1 is dropped when the first connection was terminated.

Since any changes to a declared temporary table are not logged, when a ROLLBACK statement is executed, the temporary table cannot go back to the state of the last commit point. Instead, all rows of the temporary table are deleted as if all rows were inserted during this unit of work. If the temporary table was created in this unit of work, the table will be dropped. However, if the temporary table was dropped in this unit of work, the table will be restored without any rows.

4.1.3.7 Summary tables

Summary tables provide a useful way to improve the response time of dynamic SQL queries. This type of table will typically be used to:

- Hold a frequently accessed subset of data
- Show a the result of a join between a group of tables
- Show an aggregate of data over multiple dimensions

Summary tables are mostly a lot smaller than the base fact tables from which they were created. The REFRESH option specified when the summary table is created determines when data in the table is updated. For more information on the options available, refer to the *DB2 UDB SQL Reference*, SC09-2974.

The optimizer will access a summary table if it determines that the results for the query can be satisfied by accessing data in the summary table instead of accessing the base fact tables.

Enhancements to summary tables

There are a number of extensions in Version 7.1 to help in the use and management of summary tables. In Version 7.1 a user can now:

- Use the refresh immediate option for a replicated table:
  Starting with DB2 UDB V7.1, replicated summary tables can now be created using the REFRESH IMMEDIATE option.
- Convert between a regular table (type 'T') and a summary table (type 'S'):
  This can be done using the SET SUMMARY AS clause of the ALTER TABLE statement. See the *DB2 UDB SQL Reference*, SC09-2974 for more detail.
- Refresh multiple tables concurrently:
  You can specify multiple summary tables in a REFRESH TABLE statement and the updates will take place concurrently.

4.1.4 Table locks

Tables are one of the database objects that can be explicitly locked, others being a database or table space. All other objects are implicitly locked, like rows.

4.1.4.1 LOCKSIZE

LOCKSIZE is a table parameter that can be changed using the ALTER TABLE statement. In DB2, tables use row locking by default; by using the LOCKSIZE parameter you can override this. The options available allow you to set default locking to be either row or table. The example below shows how to set table level locking as the default.

```
ALTER TABLE db2inst1.table1 LOCKSIZE TABLE
```

In terms of applications, the previous ALTER TABLE statement would mean that a share or exclusive lock would be acquired on the table; intent locks (except intent none) are not used. By using this value you may improve the performance of queries simply by limiting the number of locks that need to be acquired. Do not forget, though, that the trade-off is concurrency; this could be reduced, since all locks will be held over the whole table.

Use of this option in the table definition will not prevent normal lock escalation from occurring.

4.1.4.2 LOCK TABLE or ALTER TABLE?

Like the LOCKSIZE parameter of the ALTER TABLE statement, the LOCK TABLE statement can also be used to override the way locks are acquired on tables. Howeve, when the LOCK TABLE statement is issued, the table is locked only until the unit of work commits or is rolled back. The two locking modes available are SHARE and EXCLUSIVE.

- Share mode:
  In share mode, concurrent applications can only issue read-only operations on the selected table.
- Exclusive mode:
  In exclusive mode, concurrent applications cannot perform any operations against the selected table. One exception to this is a situation where application processes are using isolation level Uncommitted Read (UR), which can execute read-only operations.

4.1.5 Recovering dropped tables

Recoverability of individual tables can also impact performance. What we mean here is the ability to be able to recover a table quickly in the event that it was dropped accidentally.

Normally, the only way to recover a table is to restore from a database backup, which, from a performance point of view, may not be desirable if your database is very large and you have only dropped one table.

It is worth considering which tables in your database are the most important — for example, those whose data is critical. For those tables, it might be worth utilizing the dropped table recovery feature. This allows you to recover a table by performing only a table space level restore and then rolling forward through the logs. An obvious advantage here is speed and availability, because the database is still accessible to users.

This functionality is enabled at the table space level and applies to regular table spaces only. To find out which table spaces have this feature enabled, you can run the following command when connected to your database:

```
SELECT TBSPACE,DROP_RECOVERY FROM SYSCAT.TABLESPACES
```

The `DROP_RECOVERY` column will indicate if this feature is enabled (Y) or disabled (N). To enable the feature for a particular table space, run the following command when connected to the database:

```
ALTER TABLESPACE USERSPACE1 DROPPED TABLE RECOVERY ON
```

For more information on this, please refer to the *DB2 UDB Administration Guide: Implementation*, SC09-2944.

Once you have created your tables, it is worth extracting a copy of the DDL for the tables in your database. This is good for reference and to help recreate the table structure; for example, accidentally dropping a table when you have no database backup. The DDL can be extracted using the `db2look` command:

```
db2look -d <databasename> -a -e -o <outputfile> -m
```

See the *DB2 UDB Command Reference,* SC09-2951 for details on `db2look` options.

4.2 Indexes

An index is a list of the physical locations of rows sorted by the contents of one or more specified columns of the base table. You can create indexes in order to:

- Avoid unnecessary table scans
- Ensure uniqueness
- Provide ordering
- Avoid sorts
- Speed up frequently executed queries
- Speed up join predicates and support referential integrity

Indexes can reduce access time significantly; however, indexes can also have adverse effects on performance. Before creating indexes, consider the effects of multiple indexes on disk space and processing time:

- Each index takes up a certain amount of storage or disk space. The exact amount is dependent on the size of the table and the size and number of columns included in the index.
- Each INSERT or DELETE operation performed on a table requires additional updating of each index on that table. This is also true for each UPDATE operation that changes an index key.

Because of this trade-off (improved query performance versus longer insert/delete/update times), you should take different indexing strategies based on the type of workload that your database is tasked to perform. For example, you should create fewer indexes on the OLTP tables on which the insert/update rate is high, than on the DSS tables where queries predominate.

When you create indexes, there are a number of options available to you, which affect its performance. Here we will look at those which affect layout.

4.2.1 Separate table space for indexes?

One of the main reasons for separating your indexes from the data by placing them in a separate table space is that you may wish to place them on faster devices or you may wish to allocate them their own buffer pool.

However, before you do this, you should see the other recommendations on when to separate data and indexes provided in 3.3.9.4, “Considerations for separating data and indexes” on page 70.

4.2.2 Free space

Index free space can be specified explicitly in the CREATE INDEX statement with the use of PCTFREE. The value specified will used as the percentage of each index leaf page to leave as free space. For non-leaf pages, the minimum of the amount explicitly specified and 10% will be reserved.

In the following cases, you should choose a smaller value for PCTFREE, which will save space and reduce index I/Os:

- The index is never updated.
- The index entries are in ascending order and mostly high-key values are inserted into the index.
- The index entries are in descending order and mostly low-key values are inserted into the index.

If you have indexes which get updated frequently, then a larger value would be recommended to avoid index page splits.

Note

Page splits reduce performance because they result in index pages no longer being sequential or contiguous. The resulting effect is that the ability to perform index page prefetching is reduced.

Specifying a large value will also help ensure that there is enough free space to maintain a clustered index if you have defined the index as such. With clustering indexes DB2 attempts to keep the data in the same order as the order specified by the index.

If a value of PCTFREE 25 was used, for example, then 25% of each index leaf page would be kept as free space during initial index build or rebuild (as a result of a load or reorg). PCTFREE can therefore help reduce the frequency of index page splits, which will reduce performance. Usage of PCTFREE can also reduce the need to reorganize the index.

Note

An index can be reorganized by dropping it and recreating it, or by using the REORG utility. The REORG utility is costly, but does ensure that index pages are clustered, which benefits index scans.

4.2.2.1 On-line index reorg

Related to the PTCFREE clause of the CREATE INDEX statement is another clause MINPCTUSED. This acts as a threshold for the minimum amount of used space on an index leaf page. When free space on an index page hits the MINPCTUSED value, for example, after an index key is deleted from an index leaf page, the database manager attempts to merge the remaining keys with neighboring pages. This mechanism is known as an on-line index reorg. An on-line index reorg can improve space reuse and consequently reduce index I/Os. This only happens when there is sufficient space on the neighboring page to allow the merge to complete successfully. The empty page is then deleted.

Because enabling on-line index reorg causes the performance cost of checking for space to merge each time a key deletion occurs, you should enable on-line index reorg for an index when this overhead can be justified by the benefit of better space utilization for the index.

The MINPCTUSED value should be set to less than one hundred (100). This value becomes the reorganization threshold. We recommended a value for MINPCTUSED which is less than 50 percent, since goal is to merge two neighboring index leaf pages.

Index leaf pages freed following an on-line index reorg are available for re-use. The restriction is that freed pages are only available to other indexes in the same table.

Index non-leaf pages will not be freed following on-line index reorg. Full table reorg will make the index as small as possible. Leaf and non-leaf pages will be reduced in number, as will index levels.

If MINPCTUSED has a value of zero, on-line reorg of that index will not be performed. Existing indexes with a zero value that require this feature will have to be dropped and recreated to enable it. The value of MINPCTUSED can be obtained from the SYSCAT.INDEXES system catalog table using the following statement:

```
SELECT INDNAME,MINPCTUSED FROM SYSCAT.INDEXES
```

4.2.3 Include columns

We mention the INCLUDE clause of the CREATE UNIQUE INDEX statement, as it can be used to increase the number of queries eligible for index-only access. Index-only access improves the performance of the queries, since the base table does not need to be accessed, and typically the base table size is much larger than the index.

You can use this clause to specify additional columns to be used as part of the index but they are not part of the unique index key. When applications issue queries, the include columns can be specified as regular index keys.

The only restrictions are that the columns selected must be distinct from those columns used to enforce uniqueness, and the total select cannot be more than 16 columns (the sum of which cannot be greater than 255 bytes).

4.2.4 Clustering indexes

The `CLUSTER` option of the `CREATE INDEX` statement specifies that the index is the clustering index of the table. When using a clustered index, DB2 UDB attempts to insert data using the index to find the location of the same or next key, and then attempts to insert the data on the same page. By clustering the data in this way, we improve the performance of operations such as prefetch and range scans. Only one clustered index is allowed per table.

We can use a clustered index to try to optimize queries that retrieve multiple records in index order. The result of clustering should be that fewer physical I/Os are performed. When a clustered index is defined after a table exists, then use `ALTER TABLE PCTFREE <value>` before performing any new `LOAD` or `REORG` operations. By doing this, you will ensure that the percentage value specified for `PCTFREE` will be left on each page during the `LOAD` or `REORG`. See the `ALTER TABLE` statement in the *DB2 UDB SQL Reference*, SC09-2974 for more information. By allocating this extra free space, we increase the probability that the cluster insert algorithm can find free space on the desired page.

If you perform updates that considerably enlarge records, then we may get pointer overflow records which degrade performance, as we then have rows which no longer fit in their original data pages.

Note

A clustered index may not be defined for a table which is set to use APPEND MODE. See the 4.1, "Tables and performance" on page 85 for more information on `APPEND MODE`.

4.2.5 Index Advisor Wizard

The Index Advisor Wizard will determine the best set of indexes for a given workload. A workload contains a set of weighted SQL statements that can include queries as well as updates. The wizard will recommend which new indexes to create, which existing indexes to keep, and which existing indexes to drop.

When the index advisor recommends indexes, these are not immediately created; instead they are stored in an Explain table called `ADVISE_INDEX`. Information can also be inserted into this table using the `db2advis` tool, which we will look at shortly or manually using SQL.

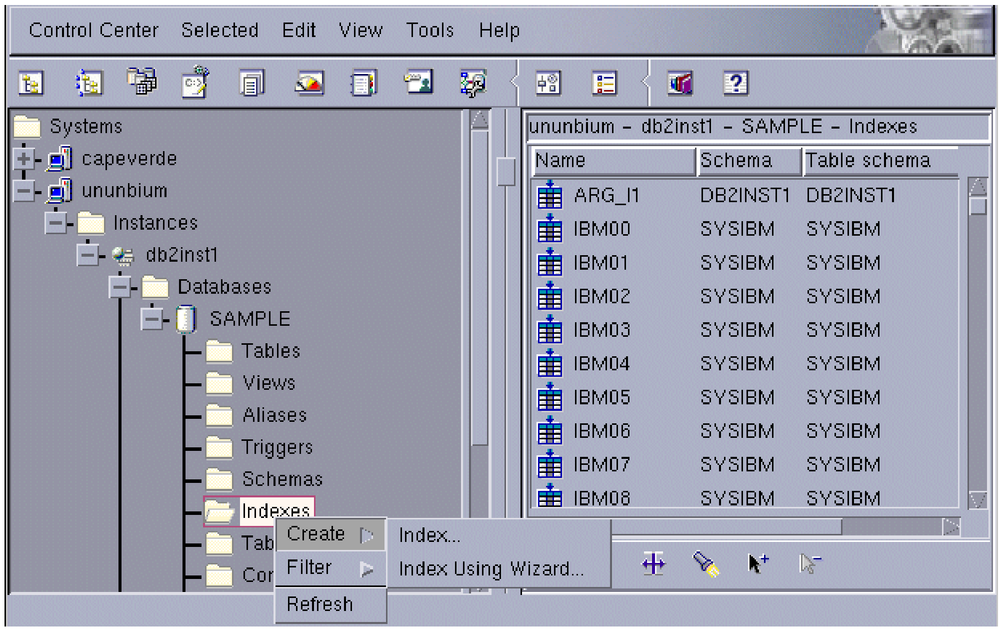
The workload information which is considered by the Index Advisor Wizard is stored in an additional Explain table `ADVISE_WORKLOAD`.

4.2.5.1 Using the Index Advisor Wizard

The Index Advisor Wizard can be invoked from the DB2 UDB Control Center or by using the `db2advis` utility.

Control Center

To start the Index Advisor Wizard from the Control Center, you must first highlight the Indexes folder for your database and then right-click to see the Create menu as in Figure 20 below.



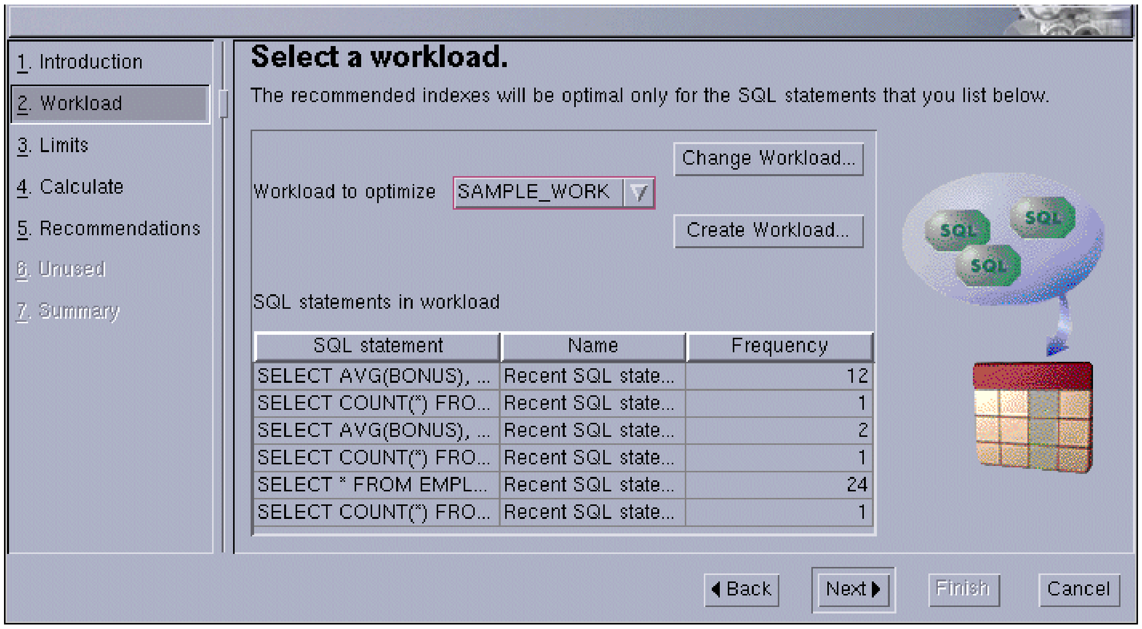
ununbium - db2inst1 - SAMPLE - Indexes

Name	Schema	Table schema
ARG_I1	DB2INST1	DB2INST1
IBM00	SYSIBM	SYSIBM
IBM01	SYSIBM	SYSIBM
IBM02	SYSIBM	SYSIBM
IBM03	SYSIBM	SYSIBM
IBM04	SYSIBM	SYSIBM
IBM05	SYSIBM	SYSIBM
IBM06	SYSIBM	SYSIBM
IBM07	SYSIBM	SYSIBM
IBM08	SYSIBM	SYSIBM

Figure 20. Index Advisor Wizard — starting from control center

In Figure 20, notice that by selecting the Indexes folder and right-clicking, we can see the option, **Create**. From here, we choose the **Index Using Wizard** option.

We then see a screen like that shown in Figure 21. Here we define the workload to be used by the Index Advisor Wizard, so that recommendations can be made on what indexes we could create to minimize the total workload cost. As you can see in Figure 21, a workload is a selection of SQL statements which the database would normally have to process over a given period. Statements in the workload can be added or removed as desired. More than one workload definition can be created; each is stored in the `ADVISE_WORKLOAD` table.



Select a workload.

The recommended indexes will be optimal only for the SQL statements that you list below.

Workload to optimize: SAMPLE_WORK

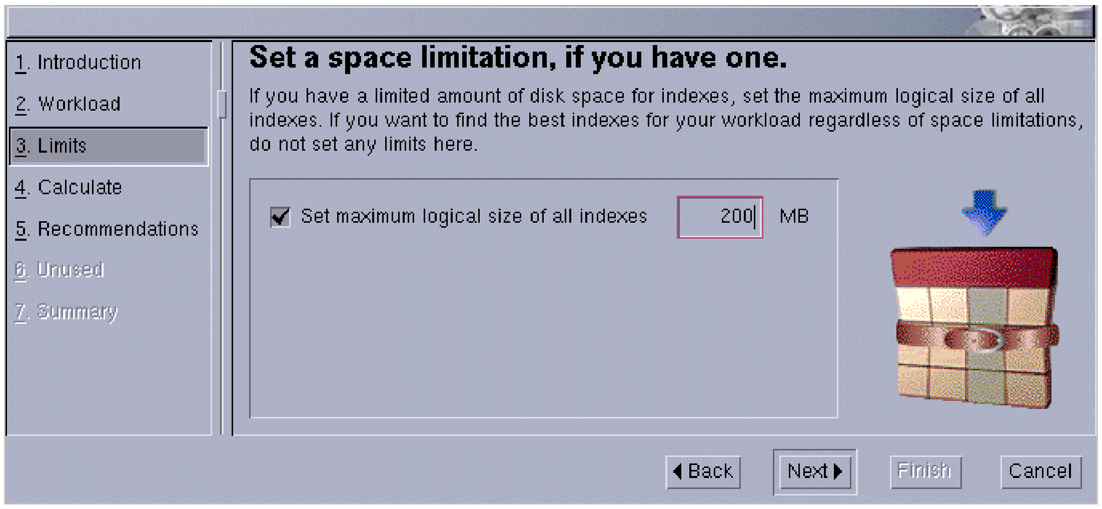
SQL statements in workload

SQL statement	Name	Frequency
SELECT AVG(BONUS), ...	Recent SQL state...	12
SELECT COUNT(*) FRO...	Recent SQL state...	1
SELECT AVG(BONUS), ...	Recent SQL state...	2
SELECT COUNT(*) FRO...	Recent SQL state...	1
SELECT * FROM EMPL...	Recent SQL state...	24
SELECT COUNT(*) FRO...	Recent SQL state...	1

Figure 21. Index Advisor Wizard — defining workload

The next screen shown in Figure 22 allows the user to set a specific value for the maximum disk space you wish to allocate for any new indexes. It is recommended that a value be specified if disk space is scarce.

If you decide to set a disk space limit, be aware that the indexes recommended may not be the most optimal for your workload, as the Index Advisor Wizard may discard some indexes because they would exceed the available disk space limit. If you want to see what could be created regardless of disk space, do not set a limit.



Set a space limitation, if you have one.

If you have a limited amount of disk space for indexes, set the maximum logical size of all indexes. If you want to find the best indexes for your workload regardless of space limitations, do not set any limits here.

[✓] Set maximum logical size of all indexes 200 MB

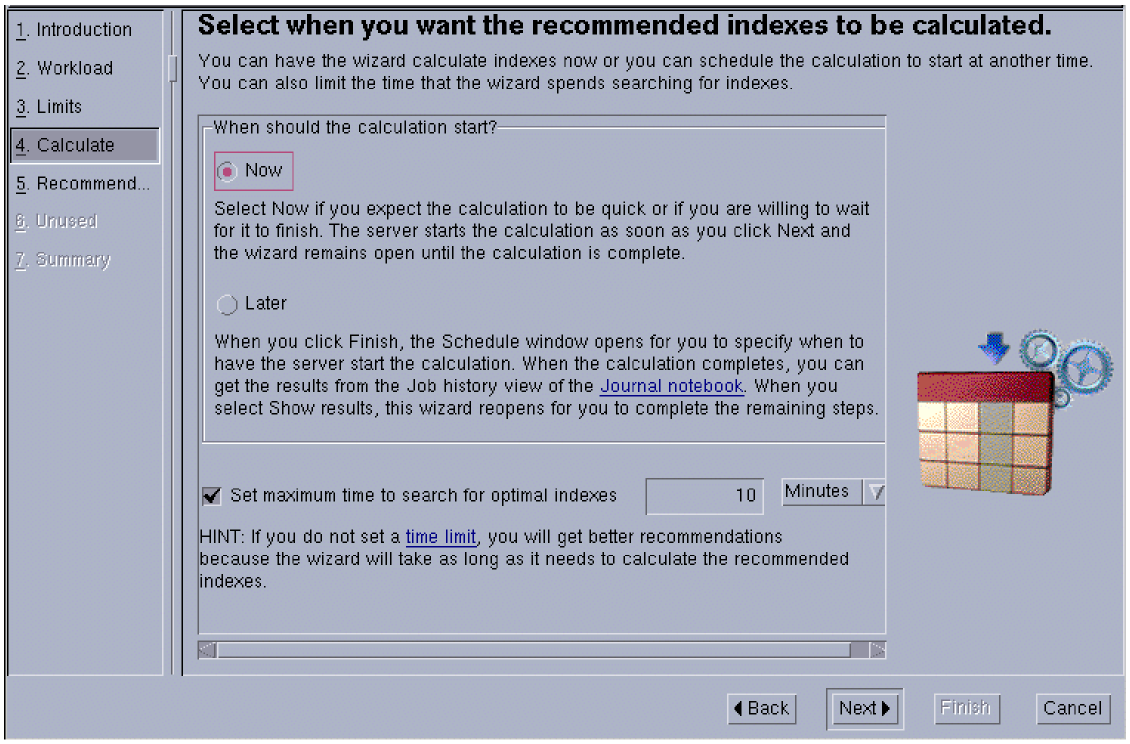
Figure 22. Index Advisor Wizard — setting index space limit

If your workload consists of many complex queries, then it may take some time to calculate the recommended indexes. The screen shown in Figure 23 can be used to determine not only when the calculations will be performed, but also the maximum amount of time they are allowed to run for.

By using a scheduled time, you can allow the index advisor to run at a time when database resources are not being fully utilized (such as at night or on a weekend). You can also limit the length of time the wizard can run, thereby limiting execution time to a set window.

If you do specify a time limit in which the wizard can run, then the index recommendations made may not be the most optimal, as the wizard may not have been allowed the time required to calculate the best indexes for your workload.

However, if the results are returned before the time limit, then assume that these are the most optimal recommendations that the wizard was able to make.



Select when you want the recommended indexes to be calculated.

You can have the wizard calculate indexes now or you can schedule the calculation to start at another time. You can also limit the time that the wizard spends searching for indexes.

When should the calculation start?

[✓ Now]

Select Now if you expect the calculation to be quick or if you are willing to wait for it to finish. The server starts the calculation as soon as you click Next and the wizard remains open until the calculation is complete.

Later

When you click Finish, the Schedule window opens for you to specify when to have the server start the calculation. When the calculation completes, you can get the results from the Job history view of the Journal notebook. When you select Show results, this wizard reopens for you to complete the remaining steps.

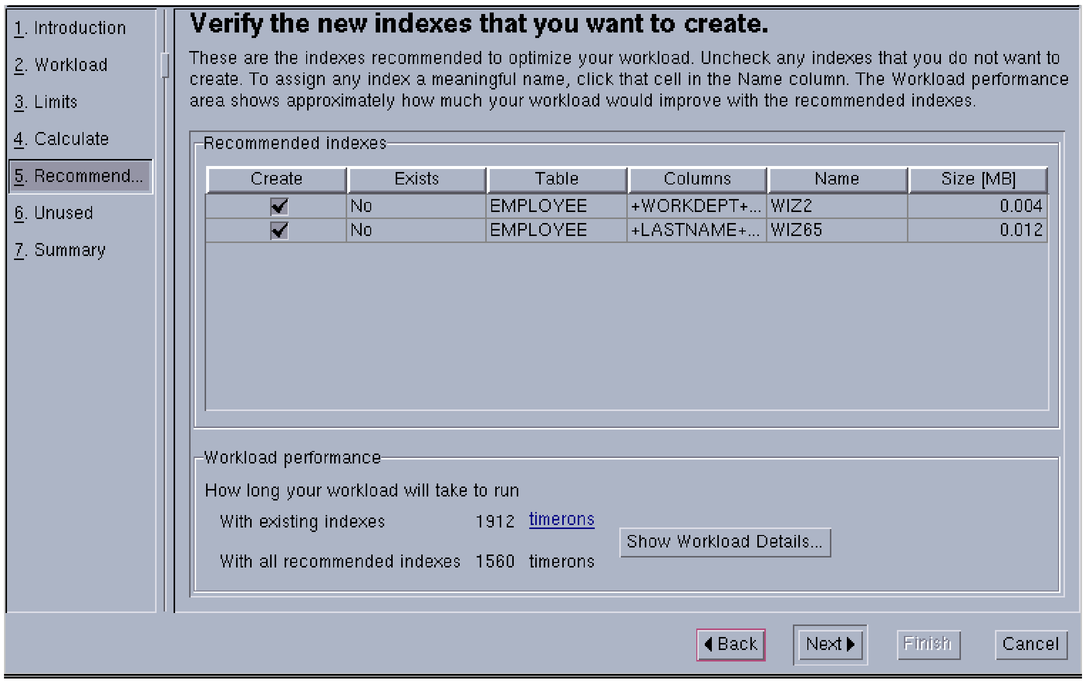
[✓] Set maximum time to search for optimal indexes 10 Minutes

HINT: If you do not set a time limit, you will get better recommendations because the wizard will take as long as it needs to calculate the recommended indexes.

Figure 23. Index Advisor Wizard — select calculation time

Once the Index Advisor Wizard has completed, you will see a screen similar to that shown in Figure 24. We can see here that the wizard has recommended that two indexes could be created to help minimize the overall cost of this workload on the databases resources. For each index, we can see the table the index was created against, what columns were used to create the index, and what the disk space requirements to hold the index will be. We can also change the index name from this screen.

At the bottom of the screen is a section called Workload performance, which shows the improvement in workload duration time, if the new indexes are created. The values show are measured in timerons. Timerons do not correlate directly to any specific unit of elapsed time, but provide an estimate of the resources required by the database manager to execute the queries within the workload. Resources would include CPU (instructions required) and I/O (page transfers and seeks required).



Verify the new indexes that you want to create.

These are the indexes recommended to optimize your workload. Uncheck any indexes that you do not want to create. To assign any index a meaningful name, click that cell in the Name column. The Workload performance area shows approximately how much your workload would improve with the recommended indexes.

Recommended indexes

Create	Exists	Table	Columns	Name	Size [MB]
✓	No	EMPLOYEE	+WORKDEPT+...	WIZ2	0.004
✓	No	EMPLOYEE	+LASTNAME+...	WIZ65	0.012

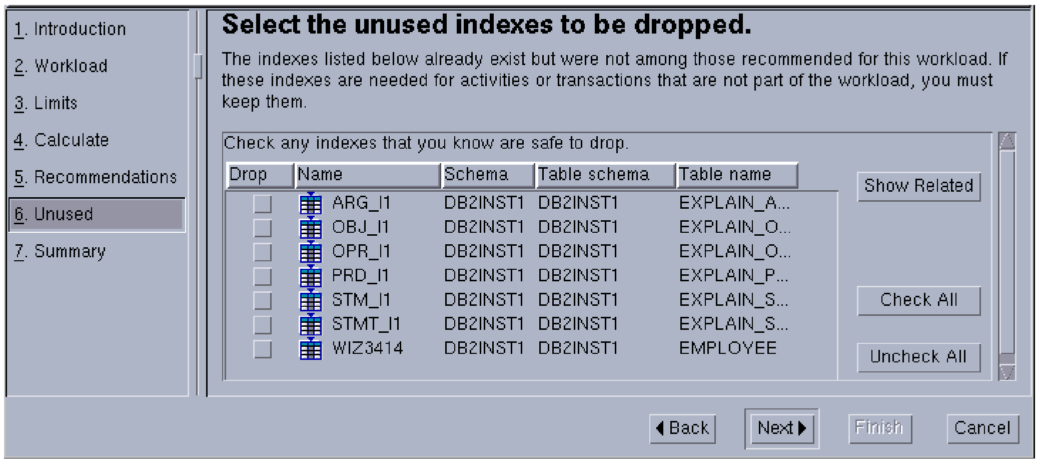
Workload performance

How long your workload will take to run

With existing indexes	1912	timerons
With all recommended indexes	1560	timerons

Figure 24. Index Advisor Wizard — choosing indexes to create

In Figure 25, we can see which indexes the wizard flagged as being unused during the execution of the workload. Do not remove indexes unless you are sure that they are not required by other applications or workloads.



Select the unused indexes to be dropped.

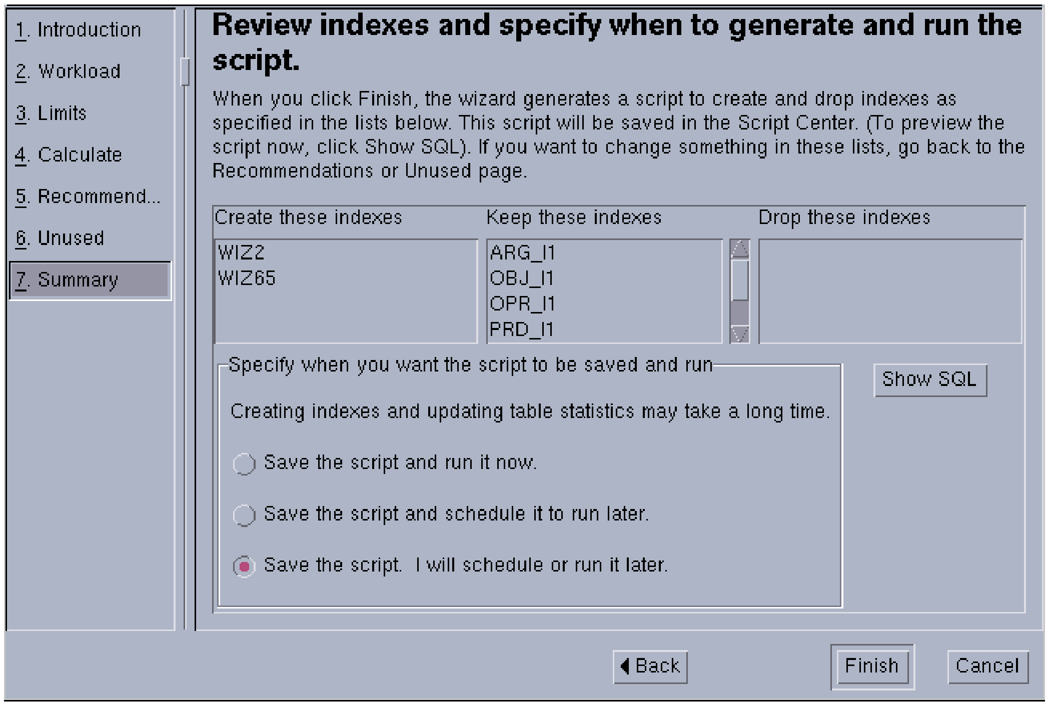
The indexes listed below already exist but were not among those recommended for this workload. If these indexes are needed for activities or transactions that are not part of the workload, you must keep them.

Check any indexes that you know are safe to drop.

Drop	Name	Schema	Table schema	Table name
☐	ARG_I1	DB2INST1	DB2INST1	EXPLAIN_A...
☐	OBJ_I1	DB2INST1	DB2INST1	EXPLAIN_O...
☐	OPR_I1	DB2INST1	DB2INST1	EXPLAIN_O...
☐	PRD_I1	DB2INST1	DB2INST1	EXPLAIN_P...
☐	STM_I1	DB2INST1	DB2INST1	EXPLAIN_S...
☐	STMT_I1	DB2INST1	DB2INST1	EXPLAIN_S...
☐	WIZ3414	DB2INST1	DB2INST1	EMPLOYEE

Figure 25. Index Advisor Wizard — choosing indexes to remove

The results screen, Figure 26, shows a summary of the selections we have made; these include which indexes to create, existing indexes to keep, and indexes to remove. At this point you can save the actions to a script (shell script or DB2 script), and also carry them out immediately or schedule them to run later.



Review indexes and specify when to generate and run the script.

When you click Finish, the wizard generates a script to create and drop indexes as specified in the lists below. This script will be saved in the Script Center. (To preview the script now, click Show SQL). If you want to change something in these lists, go back to the Recommendations or Unused page.

Create these indexes	Keep these indexes	Drop these indexes
WIZ2	ARG_I1	
WIZ65	OBJ_I1	
	OPR_I1	
	PRD_I1	

Specify when you want the script to be saved and run

Creating indexes and updating table statistics may take a long time.

○ Save the script and run it now.

○ Save the script and schedule it to run later.

[✓ Save the script. I will schedule or run it later.]

Figure 26. Index Advisor Wizard — review results

The db2advis utility

You can use the `db2advis` utility from the command line. It works in the same way as the Index Advisor Wizard. For detailed information on running the `db2advis` utility, refer to the *DB2 UDB Command Reference*, SC09-2951.

Note

Before running the Index Advisor Wizard or the db2advis utility against a database, ensure that the Explain tables have been created for that database. Run `db2 -tf EXPLAIN.DDL` against the database to create these tables. `EXPLAIN.DDL` is found in the sqllib/misc directory in the instance owner's home directory

4.2.6 Other performance tips for indexes

Here are some other performance tips for indexes:

- Execute `RUNSTATS` utility after creating indexes.

  Whenever you create new indexes, do not forget to execute the `RUNSTATS` utility to update the statistics information after the index creation. This provides the optimizer with accurate information on the indexes and makes the optimizer determine the best access plan.

- Choose a suitable configuration (`SORTHEAP` and buffer pool size) for index creation.

  When tuning index creation performance, the amount of memory dedicated to the sorting of index keys is controlled by the `SORTHEAP` database configuration parameter. If an index is so large that it cannot be sorted in memory, a sort spill occurs. That is, the data is divided among several "sort runs" and stored in a temporary table space that will be merged later. For the optimal performance of the index creation, you should avoid a sort spill by tuning the `SORTHEAP` database configuration parameter.

- The buffer pool size is also the important factor of the index creation performance.

  A index creation needs to retrieve data pages for the index keys from the base table, and also build and save index pages. To minimize the amount of disk storage I/O caused by these data pages reading and index pages writing, the buffer pool for the table space to which the base table and indexes belong should be large enough. If the index key sort causes a sort spill, it is also important that the buffer pool for temporary table spaces be large enough to minimize the amount of disk I/O that spilling causes.

- Build indexes as a part of the load operation.

  If you are trying to load data into an empty table (or a table with few rows) and intend to create indexes on the table, you should create indexes first, then execute the `LOAD` utility. The `CREATE INDEX` statements will be completed in a second since the table is empty, then the `LOAD` utility will build the indexes as part of the load operation. This is more efficient than doing the index creation separately after the load is done, because the index keys do not have to be retrieved from the base table. If doing the indexes creation separately, each `CREATE INDEX` statement must scan the base table and retrieve the index keys.

When you perform the `LOAD` utility using the `REPLACE` option, you can specify the `STATISTICS YES` option and collect the statistics information of the base table and indexes on it. This is a faster alternative than executing the `RUNSTATS` utility after the `LOAD` statement is completed.

- Try to keep the index key size as small as possible.

  Remember that each `INSERT`, `DELETE` or `UPDATE` operation which changes an index key performed on a table requires additional updating of each index on that table. If the index key size is bigger, then this overhead is also bigger.

- Index creation is improved using intra-parallelism.

  If the database is on an SMP machine, consider setting the `INTRA_PARALLEL` database manager configuration parameter to `YES` or `SYSTEM` to take advantage of parallel performance improvements. Multiple processors can be used to scan and sort data for index creation.

4.3 64-bit engine

Starting with the first FixPak of the DB2 UDB V7.1, a new functionality, the 64-bit database engine, will be introduced. Taking advantage of a 64-bit address space will allow for the creation of:

- Larger buffer pools
- Larger sort heap
- Larger package caches
- Other resources that can consume large amounts of memory
- Utilization of additional memory that could improve performance by reducing I/O operations
- UDF enhancement

AIX and Solaris are the systems for which 64-bit DB2 UDB support is initially targeted. For other operating systems, 64-bit support will follow.

4.3.1 Libraries

In addition to the specific advantages highlighted above, a secondary but very important consideration is that, in order to allow DB2 application developers to use a 64-bit address space, they need to have available to them 64-bit DB2 libraries. This is due to the fact that, although 64-bit platforms allow both 32-bit and 64-bit processes to co-exist, they do not allow intermixing of 32-bit and 64-bit executables and libraries within the same process.

Chapter 5. Monitoring tools and utilities

Understanding your database and applications helps you tune your database and improve its performance. To do this, you may need to monitor your database system to gather performance and statistical information about the operations and tasks that are executing.

This chapter describes how to monitor the system resource usage using AIX commands such as `iostat` and `vmstat`. It also discusses the different methods available to monitor the database activity, using tools such as the Snapshot Monitor, the Event Monitor, the Performance Monitor, and the CLI/ODBC/JDBC Trace Tool. Based on the information provided by these commands and tools, you can make informed decisions about what actions need to be taken to tune the database environment.

We have developed some sample shell scripts that execute these commands and tools, and extract the useful information from the output. These scripts are shown in Appendix A, “Sample scripts” on page 335. You can use them as we show in this chapter, or modify them appropriately for your environment.

5.1 Measuring and monitoring

The results obtained when testing or benchmarking can be split into two different categories:

- Attributes being measured

  Usually, the response time is the only measured attribute. But any other metric, such as a throughput, may be the target.

- Attributes being monitored

  These attributes show the behavior of the database during the test. For example, you may be interested in the disk I/O, memory or CPU activity. AIX provides a variety of commands and tools to monitor each of the system resource activities. From the DB2 perspective, you may be interested in the way pages are being accessed through prefetchers, or the watermark of a heap left by the execution of a test. DB2 provides a variety of tools for this purpose.

When measuring and monitoring, each step of the test should be carefully considered. The method you are going to use to measure must be clearly established. You should be aware that the measuring and monitoring activity may have some affect on the results obtained. Previous steps of the test, or previous tests, may also have an impact, because pools and caches are not flushed unless you take steps to guarantee a clean measure.

Take measurements while the system is active and performance problems are evident. If no performance problems are currently evident, you may investigate potential problems and take some initial measurements to document a “stable” state. You may then set up some monitoring to be performed automatically, or instruct an administrator or user to take measurements while you are away. Keep in mind that Event and Snapshot Monitors do take up their own resources, and should be turned off after the tuning process is complete.

5.2 Maintaining tuning information

Reports of the results should be kept for future performance references. Performance is a relative issue, and progress is best tracked based on previous performance issues. It makes sense to define what information should be kept in a performance report. Reports should include not only the results, but also the configuration parameters, the environment, and values of other monitored attributes.

As mentioned at the beginning of this chapter, we have written some sample shell scripts which execute AIX commands or DB2 monitoring tools. These scripts save the output to files in the directory structure which we believe is helpful in maintaining order. See Appendix A, “Sample scripts” on page 335 for the source files and the syntax of all the scripts. You can use them to maintain your monitoring report or get some ideas for your environment from these scripts. The directories will be defined in any user’s home directory, and are outlined below.

The directory structure which our sample scripts will use is shown in Table 9.

Table 9. Suggested directory structure for maintaining point-in-time results

Directory	Purpose	Stored files
~/work	Scripts used for monitoring	sqlcache.ksh mon_stmt.ksh
~/queries	Queries used to test performance or SQL statements which have been captured from packages.	1.sql 54.sql
~/states	Hold directories representing different points in time. Zero should be for the initial state, before tuning.	0-n

Directory	Purpose	Stored files
~/states/0-n	Each subdirectory should contain output files showing different values at a given point in time.	dbm_cfg.sql db_cfg_TPC.sql reorgchk_all.out
~/states/0-n/ bak	Output files are never over-copied. They are moved to here with a unique name.	dbm_cfg.sql.1234 5
~/results	Hold directories representing different points in time. Zero should be for the initial state, before tuning.	0-n
~/results/0-n	Each subdirectory should contain output files of measurements and monitors at a given point in time.	1.out, 1.sum sqlcache.out sqlcache*.sum
~/results/0-n/bak	Output files are never over-copied. They are moved to here with a unique name.	1.sum.12345
~/diff	A useful command to compare states or results is diff. Outputs should be kept here.	q2_r20_r21.diff

Most of our sample scripts have a parameter -o which accepts a number indicating the subdirectory where output should be placed under either the ~/states or ~/results directories. If the output file already exists there (the utility has been run with the same parameters), then the older file is copied to a subdirectory `bak` with a slightly different filename (the PID number will be attached to it). If any of the directories do not exist, it will be created.

Another standard which we have tried to maintain is that all of the scripts will force you to enter a comment which is placed in the beginning of the summary output (report) file; for example, "RUN BEFORE INDEX CREATION" or "FIRST EXECUTION AFTER DB2START". The run date and script parameters (invocation) are also put in the beginning of the summary output file.

The following sections show each directory in the directory structure which our sample shell scripts will use. You do not have to use our sample scripts or exactly the same directories to save monitored data. However, we recommend that you keep any performance monitoring results for tracking the system performance and also for future performance references.

5.2.1 States subdirectory

When first approaching a system which may need to be tuned, you should document the values of various setting and parameters. These may be operating system settings such as paging space, or database parameters such as those in the database configuration file.

In our set of suggested directories, a new subdirectory, whose name is a number, should be created in the ~/states directory. Before modifying any settings or parameters, all of the state-type output files should be saved to this new directory. The types of DB2 and operating system information which should be saved in this directory could include:

- Values of the database manager configuration parameters
- Values of the database configuration parameters
- Output of the `REORGCHK` (with `CURRENT STATISTICS`) command
- Current monitor switches status
- Defined and active Event Monitors
- Schema (DDL) and statistics information from `db2look`
- Existing tables and indexes (including basic statistics)
- DB2 registry variables (`db2set -all`)
- Installed software set (`lslpp -L`)
- Results of `lsps` (paging space) and any of the other UNIX monitors mentioned in this chapter

After making a modification to a setting or tunable parameter, another subdirectory for the new state, whose name is the next number, should be created under ~/states, and the file which might have been affected should be regenerated in this new directory. Modifications may include:

- Modifying the database or database manager configuration parameters
- Adding or removing indexes
- Performing `RUNSTATS` or `REORG TABLE` command
- Adding memory or modifying the paging space

The following example shows a sample SQL statement to obtain information on existing tables. You should get and save this information:

```
SELECT CAST( tbspace || '.' || tabschema || '.' || tabname AS CHAR(40))
  AS
table
, index_tbspace
, SUBSTR( CHAR(create_time), 1, 19)     AS create_time
, tableid
, type
, status
, card
, overflow
, pctfree                               AS PCFREE
, SUBSTR( CHAR(stats_time), 1, 19)      AS stats_time
FROM syscat.tables
WHERE       tabschema NOT IN ('SYSIBM', 'SYSCAT', 'SYSSTAT')
  AND       tabname NOT LIKE 'EXPLAIN_%'
  AND       tabname NOT LIKE 'ADVISE_%'
ORDER BY table;
```

The result will be like the following:

```
TABLE                                    INDEX_TBSPACE      CREATE_TIME
TABLEID TYPE STATUS CARD                 OVERFLOW    PCFREE STATS_TIME
---------------------------------------- ------------------ -------------------
------- ---- ------ -------------------- ----------- ------ -------------------
TPCDDATA.DB2INST1.CUSTOMER               TPCDINDEX          2000-05-05-19.24.31
     11 T    N                    150000           0     -1 2000-05-31-21.46.55
TPCDDATA.DB2INST1.LINEITEM               TPCDINDEX          2000-05-05-19.24.29
      7 T    N                   6001215           0     -1 2000-05-10-20.04.14
TPCDDATA.DB2INST1.NATION                 TPCDINDEX          2000-05-05-19.24.29
      8 T    N                        25           0     -1 2000-05-12-17.10.53
TPCDDATA.DB2INST1.ORDERS                 TPCDINDEX          2000-05-05-19.24.30
      9 T    N                   1500000           0     -1 2000-05-10-20.05.21
TPCDDATA.DB2INST1.PART                   TPCDINDEX          2000-05-05-19.24.27
      4 T    N                    200000           0     -1 2000-05-12-17.10.08
TPCDDATA.DB2INST1.PARTSUPP               TPCDINDEX          2000-05-05-19.24.28
      5 T    N                    800000           0     -1 2000-05-12-17.10.52
TPCDDATA.DB2INST1.REGION                 TPCDINDEX          2000-05-05-19.24.28
      6 T    N                         5           0     -1 2000-05-12-17.10.53
TPCDDATA.DB2INST1.SUPPLIER               TPCDINDEX          2000-05-05-19.24.30
     10 T    N                     10000           0     -1 2000-05-10-20.07.45

  8 record(s) selected.
```

The following example shows a sample SQL statement to obtain information on existing indexes. You should get and save this information:

```
SELECT
          CAST( tabschema || '.' || tabname AS CHAR(30))        AS table
        , indname
        , indextype
        , SUBSTR( CHAR(create_time), 1, 19)                     AS create_time
        , CAST( colnames AS CHAR(79))                           AS columns
        , TRANSLATE( CHAR(' ',30), '-', ' ')                    AS seperator
        , clusterratio
--      , clusterfactor
        , pctfree
        , uniquerule                                            AS uniquer
        , SUBSTR( CHAR(stats_time), 1, 19)                      AS stats_time
FROM
        syscat.indexes
WHERE
        tabschema != 'SYSIBM'
  AND
        tabname NOT LIKE 'EXPLAIN%'
ORDER BY
          table
        , uniquerule DESC
        , columns;
```

The result will be like the following:

```
TABLE                          INDNAME            INDEXTYPE CREATE_TIME
COLUMNS
SEPERATOR                      CLUSTERRATIO PCTFREE UNIQUER STATS_TIME
------------------------------ ------------------ --------- -------------------
-------------------------------------------------------------------------------
------------------------------ ------------ ------- ------- -------------------
DB2INST1.CUSTOMER              C_NK_CK            REG       2000-05-10-19.51.43
+C_NATIONKEY+C_CUSTKEY
------------------------------           55      -1 U       2000-05-31-21.46.55
DB2INST1.CUSTOMER              C_MS_CK            REG       2000-05-10-19.51.34
+C_MKTSEGMENT+C_CUSTKEY
------------------------------           91      -1 D       2000-05-31-21.46.55
DB2INST1.LINEITEM              L_OK               REG       2000-05-10-22.50.27
+L_ORDERKEY
------------------------------           -1      -1 D       -
DB2INST1.LINEITEM              L_PK               REG       2000-05-10-22.52.24
+L_PARTKEY
------------------------------           -1      -1 D       -
DB2INST1.LINEITEM              L_PKSKOKEPDSQN     REG       2000-05-10-22.54.24
+L_PARTKEY+L_SUPPKEY+L_ORDERKEY+L_EXTENDEDPRICE+L_DISCOUNT+L_QUANTITY
------------------------------           -1      -1 D       -
```

5.2.1.1 The db2look tool

The `db2look` tool can be used to extract the Data Definition Language (DDL) which can be used to redefine database objects. This makes it a useful tool for documenting the definitions of a database. The types of information which can be extracted include:

- Table spaces
- Buffer pools
- Tables and views
- Indexes
- Constraints
- Triggers
- Privileges
- Statistics
- Database Configuration parameters (partial)
- DB2 Registry values (partial)

See the *Command Reference*, SC09-2951 for more details about `db2look`. The parameters which offer the most complete information are:

```
db2look -d dbname -m -l -a -x -e -f
```

Sample script

We have written the sample shell script, `db2look.ksh`. This script can be used to execute `db2look` with default parameters and place the output in a `states` subdirectory. It also places a comment at the beginning of the script which is generated. The following example shows a sample invocation:

```
$ db2look.ksh -c "FIRST LOOK" -d tpc -o 1
[Created: /home/tetsur3/states/1/bak/db2look_TPC.sql.32062]
[Creating: /home/tetsur3/states/1/db2look_TPC.sql]
% Generate statistics for all creators
% Creating DDL for table(s)
% Running db2look in mimic mode
```

The source file and the complete syntax of this sample script are included in Appendix A, “Sample scripts” on page 335.

In this sample invocation, the output file will be `~/states/1/db2look_TPC.sql`. The following example shows the beginning of the generated script:

```
$ head -26 ~/states/1/db2look_TPC.sql
-- FIRST LOOK
-- ---------------------------------------
-- Invocation: db2look.ksh -c FIRST LOOK -d tpc -o 1
-- Thu Jun 1 22:22:48 PDT 2000
--
-- db2look -d TPC -m -l -a -x -e -f
--
-- This CLP file was created using DB2LOOK Version 7.1
-- Timestamp: Thu Jun  1 22:22:48 PDT 2000
-- Database Name: TPC
-- Database Manager Version: DB2/6000 Version 7.1.0
-- Database Codepage: 819


CONNECT TO TPC;

------------------------------------
-- DDL Statements for BUFFERPOOLS --
------------------------------------

CREATE BUFFERPOOL "TPCDDATABP"  SIZE 5000 PAGESIZE 8192 NOT EXTENDED STORAGE;

CREATE BUFFERPOOL "TPCDTEMPBP"  SIZE 10000 PAGESIZE 8192 NOT EXTENDED STORAGE;

CREATE BUFFERPOOL "TPCDINDEXBP"  SIZE 5000 PAGESIZE 8192 NOT EXTENDED STORAGE;
```

The upd_cfg.ksh script

Another example of the type of information which should be saved (in the ~/states directory) prior to making any tuning changes is the values of the database manager and database configuration parameters. We have written the sample script `upd_cfg.ksh`, which executes a `GET DBM CFG` or `GET DB CFG` command and save the output, at the same time generating a script file which can return all of the parameters to their current value. The generated script can be executed by using the `db2 -tvf` command. In this way, if you experiment with a number of different tunable parameters (not recommended in general), then you can always return to the values (state) which were saved in this script.

The following example shows a sample invocation of the `upd_cfg.ksh` script, and the type of script which is generated. This generated script can be executed as `db2 -tvf ~/states/1/db_cfg_TPC.sql` to restore these values:

```
$ upd_cfg.ksh -c "BEFORE CHANGES" -o 1 -d tpc
$ head -45 ~/states/1/db_cfg_TPC.sql
-- BEFORE CHANGES
-- ---------------------------------------
-- Invocation: /home/tetsur3/work/upd_cfg.ksh -c BEFORE CHANGES -o 1 -d tpc
-- Wed May 31 20:14:17 PDT 2000
--
-- db2 get db cfg for TPC
--
UPDATE DB CFG FOR TPC USING
--       Database Configuration for Database TPC
--
-- Database configuration release level                    = 0x0900
-- Database release level                                  = 0x0900
--
-- Database territory                                      = US
-- Database code page                                      = 819
-- Database code set                                       = ISO8859-1
-- Database country code                                   = 1
--
-- Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE
                                           DYN_QUERY_MGMT    DISABLE
--
-- Directory object name                    (DIR_OBJ_NAME) =
                                             DIR_OBJ_NAME    ''
-- Discovery support for this database       (DISCOVER_DB) = ENABLE
                                              DISCOVER_DB    ENABLE
--
-- Default query optimization class         (DFT_QUERYOPT) = 5
                                             DFT_QUERYOPT    5
-- Degree of parallelism                      (DFT_DEGREE) = 1
                                               DFT_DEGREE    1
-- Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO
                                          DFT_SQLMATHWARN    NO
-- Default refresh age                   (DFT_REFRESH_AGE) = 0
                                          DFT_REFRESH_AGE    0
-- Number of frequent values retained     (NUM_FREQVALUES) = 10
                                           NUM_FREQVALUES    10
-- Number of quantiles retained            (NUM_QUANTILES) = 20
                                            NUM_QUANTILES    20
--
-- Backup pending                                          = NO
--
-- Database is consistent                                  = NO
-- Rollforward pending                                     = NO
-- Restore pending                                         = NO
```

If you want to obtain the database manager configuration parameters, simply leave out the `-d` parameter. The source file and the complete syntax of this sample script are included in Appendix A, “Sample scripts” on page 335.

Note

Parameters which are calculated by default, such as `PCKCACHESZ` (which defaults to `MAXAPPLS`*8), are set to -1 to force the recalculation.

5.2.2 Queries subdirectory

As part of the tuning process, it might be necessary to experiment on a number of specific SQL statements. These queries could be provided to you, or you may have to extract them from various monitors. For example, the `sqlcache.ksh` script, which we will introduce later, extracts SQL statements from the Dynamic SQL Statement cache and places them in the `queries` directory. The queries could then be analyzed more carefully, for example, with the `db2batch` or Explain tools (discussed in this chapter).

5.2.3 Results subdirectory

The types of files which should be placed in the `~/results` subdirectories are the outputs of utilities like `db2batch`, Explain, monitors (all covered in this chapter) and some SQL queries. Most of the utilities mentioned in this chapter save their results in this subdirectory (a number, under `~/results`).

For example, the `sqlcache.ksh` utility saves the output of a Snapshot Monitor, and a summary of it as specified by parameters, in a subdirectory under `~/results`. (Optionally, it could save it in the current directory.)

5.3 AIX monitoring tools

The AIX operating system provides a variety of commands and tools to monitor the system resource activities. Here we introduce some useful commands. See Appendix A for more AIX commands.

5.3.1 Online monitor — nmon

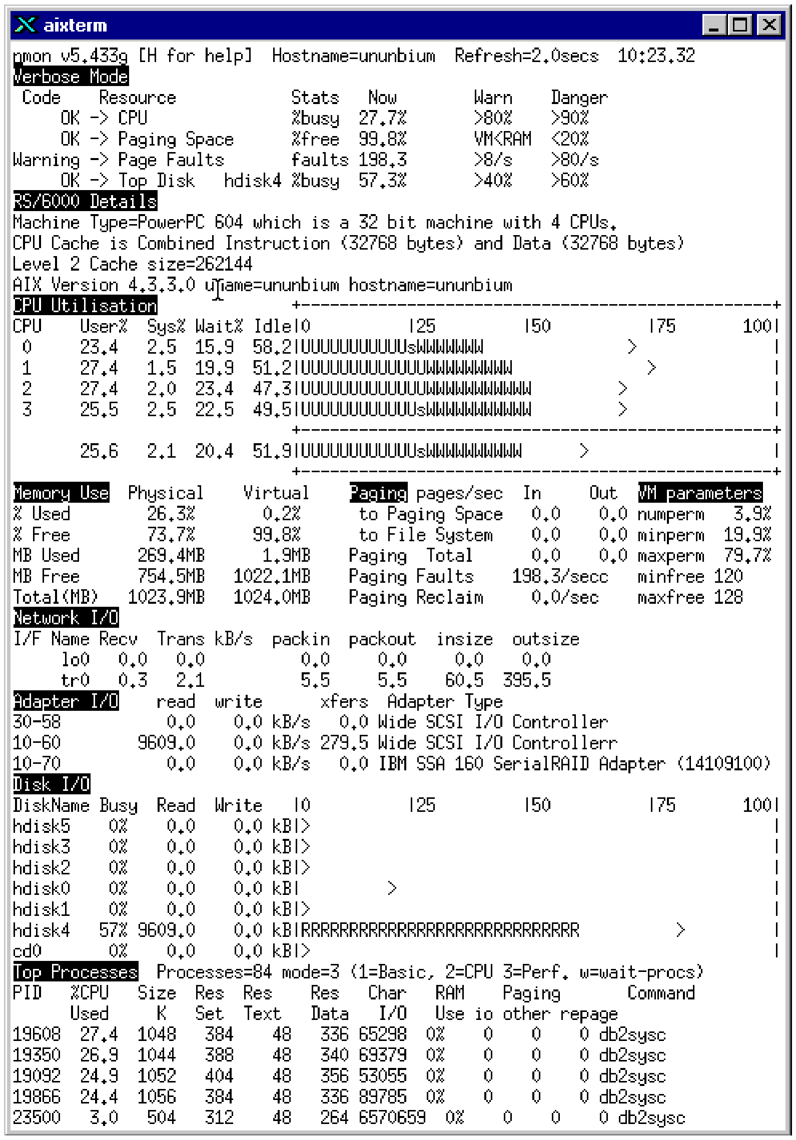
The `nmon` monitor is not included in the AIX software (or performance pack). It is available at:

`http://w3.aixncc.uk.ibm.com`

It is also available to IBM Business Partners via PartnerInfo.

As shown in Figure 27, `nmon` is a graphical tool which shows a variety of elements in real-time, such as CPU, disk, memory and process information. While it is an extremely useful tool for performing an overall on-line checkup of your system, its one drawback is that you cannot save the monitored information (actually, there is a way, but we did not find it to be too useful).

For example, once a process has completed spiking its CPU usage (and thereby coming to the top of the list), it might quickly drop down the list of processes, leaving you wondering what exactly it was that you saw. On the other hand, indicators such as CPU and Disk do have a peak marker which clearly shows the highest values which were hit during monitoring. It is also a very simple tool that can be used to display the parallel activity of multiple CPUs. However, one disadvantage is that the process list only shows the names of the parent processes.



```
nmon v5.433g [H for help]  Hostname=ununbium  Refresh=2.0secs  10:23.32
Verbose Mode
 Code    Resource            Stats   Now        Warn    Danger
     OK -> CPU               %busy  27.7%       >80%    >90%
     OK -> Paging Space      %free  99.8%       VM<RAM  <20%
Warning -> Page Faults       faults 198.3       >8/s    >80/s
     OK -> Top Disk    hdisk4 %busy  57.3%      >40%    >60%
RS/6000 Details
Machine Type=PowerPC 604 which is a 32 bit machine with 4 CPUs.
CPU Cache is Combined Instruction (32768 bytes) and Data (32768 bytes)
Level 2 Cache size=262144
AIX Version 4.3.3.0 uname=ununbium hostname=ununbium
CPU Utilisation              +-------------------------------------------------+
CPU    User%  Sys% Wait% Idle|0         |25         |50          |75       100|
 0     23.4   2.5  15.9  58.2|UUUUUUUUUUUsWWWWWWW               >              |
 1     27.4   1.5  19.9  51.2|UUUUUUUUUUUUUWWWWWWWW               >            |
 2     27.4   2.0  23.4  47.3|UUUUUUUUUUUUUWWWWWWWWWW          >               |
 3     25.5   2.5  22.5  49.5|UUUUUUUUUUUUsWWWWWWWWWW          >               |
                             +-------------------------------------------------+
       25.6   2.1  20.4  51.9|UUUUUUUUUUUUsWWWWWWWWW       >                   |
                             +-------------------------------------------------+
Memory Use  Physical    Virtual    Paging pages/sec  In     Out  VM parameters
% Used        26.3%       0.2%      to Paging Space   0.0    0.0 numperm   3.9%
% Free        73.7%      99.8%      to File System    0.0    0.0 minperm  19.9%
MB Used      269.4MB      1.9MB    Paging  Total      0.0    0.0 maxperm  79.7%
MB Free      754.5MB   1022.1MB    Paging Faults    198.3/secc   minfree 120
Total(MB)   1023.9MB   1024.0MB    Paging Reclaim     0.0/sec    maxfree 128
Network I/O
I/F Name Recv  Trans kB/s  packin  packout  insize  outsize
     lo0   0.0   0.0          0.0     0.0     0.0    0.0
     tr0   0.3   2.1          5.5     5.5    60.5  395.5
Adapter I/O    read  write      xfers  Adapter Type
30-58           0.0    0.0 kB/s   0.0 Wide SCSI I/O Controller
10-60        9609.0    0.0 kB/s 279.5 Wide SCSI I/O Controllerr
10-70           0.0    0.0 kB/s   0.0 IBM SSA 160 SerialRAID Adapter (14109100)
Disk I/O
DiskName Busy  Read  Write   |0          |25         |50          |75       100|
hdisk5    0%    0.0    0.0 kB|>                                                |
hdisk3    0%    0.0    0.0 kB|>                                                |
hdisk2    0%    0.0    0.0 kB|>                                                |
hdisk0    0%    0.0    0.0 kB|         >                                       |
hdisk1    0%    0.0    0.0 kB|>                                                |
hdisk4   57% 9609.0    0.0 kB|RRRRRRRRRRRRRRRRRRRRRRRRRRRRR          >         |
cd0       0%    0.0    0.0 kB|>                                                |
Top Processes  Processes=84 mode=3 (1=Basic, 2=CPU 3=Perf, w=wait-procs)
PID   %CPU   Size  Res  Res    Res   Char   RAM    Paging       Command
      Used     K   Set  Text   Data   I/O   Use io other repage
19608  27.4   1048  384    48   336 65298  0%    0    0    0 db2sysc
19350  26.9   1044  388    48   340 69379  0%    0    0    0 db2sysc
19092  24.9   1052  404    48   356 53055  0%    0    0    0 db2sysc
19866  24.4   1056  384    48   336 89785  0%    0    0    0 db2sysc
23500   3.0    504  312    48   264 6570659  0%    0    0    0 db2sysc
```

Figure 27. Sample of nmon with all elements displayed

During the remainder of this chapter, we will not discuss using `nmon`. Instead, we will concentrate on non-graphical monitoring tools whose output can be saved and summarized into files and placed in the `results` directory.

5.3.2 Virtual memory statistics — vmstat

You can use the `vmstat` command to report statistics about kernel threads in the run and wait queues, memory, paging, disks, interrupts, system calls, context switches, and CPU activity. If the `vmstat` command is used without any options, or only with the interval (and optionally, the count parameter, like `vmstat 2`), then the first line of numbers is an average since system reboot.

The following example shows a sample invocation of the `vmstat` command. The output is 10 lines with 2 seconds between outputs.

```
# vmstat 2 10
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 44720 192928   0   0   0   0    0   0 103   66  11  0  0 99  0
 1  1 44720 192928   0   0   0   0    0   0 785 1133 1115 34  3 39 24
 0  1 44720 192928   0   0   0   0    0   0 770 1156 1085 33  2 40 26
 0  1 44720 192928   0   0   0   0    0   0 762 1054 1040 32  2 38 28
 0  1 44720 192928   0   0   0   0    0   0 773 1057 1078 33  2 41 24
 0  1 44720 192928   0   0   0   0    0   0 757 1068 1042 33  2 40 25
 1  1 44720 192928   0   0   0   0    0   0 773 1072 1053 34  2 39 26
 1  1 44720 192928   0   0   0   0    0   0 777 1063 1065 33  1 39 26
 2  1 44720 192928   0   0   0   0    0   0 755 1050 1034 32  2 43 24
 4  1 44720 192928   0   0   0   0    0   0 760 1090 1056 34  2 41 23
```

5.3.3 Disk I/O statistics — iostat

The `iostat` tool is installed with the base operating system (AIX). It is useful for displaying basic average CPU statistics (although it only shows the average of all CPUs, without breaking them down) and disk transfers for each physical drive. It is enough to provide it with one parameter, which is the reporting interval. You can then stop it by pressing "Ctrl-C", or optionally specify the number of outputs. Note that the first entry of `iostat` should be ignored, as it gives the cumulative values since the last boot. The following example shows a sample invocation of the `vmstat` command:

```
# iostat 2

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.1         14.5               7.6      0.2       92.2       0.1

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           0.0       0.2       0.0      60652     23460
hdisk0           0.2       1.1       0.2     170797    465727
hdisk2           0.0       0.0       0.0          0         0
hdisk3           0.0       0.0       0.0          0         0
hdisk5           0.0       0.0       0.0          0         0
hdisk4           0.2      38.4       1.0   21321125     88328
cd0              0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0        333.6               0.0      0.2       99.8       0.0

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           0.0       0.0       0.0          0         0
hdisk0           0.0       0.0       0.0          0         0
hdisk2           0.0       0.0       0.0          0         0
hdisk3           0.0       0.0       0.0          0         0
hdisk5           0.0       0.0       0.0          0         0
hdisk4           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.5        352.1               2.5      3.2       75.2      19.1

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           1.5      20.0       2.0          0        40
hdisk0           2.5      11.5       3.0          0        23
hdisk2           0.0       0.0       0.0          0         0
hdisk3           0.0       0.0       0.0          0         0
hdisk5           0.0       0.0       0.0          0         0
hdisk4          55.9     273.7      68.4         16       532
cd0              0.0       0.0       0.0          0         0
```

5.3.3.1 iostat.ksh script

Because `iostat` displays a little more information than we want, and because the list of disk activity is listed vertically, we have written a script which traps `iostat` output and displays only part of the information horizontally, making it easier to track changes in activity. The output of the `iostat.ksh` script is saved under the `results` directory. Following is an example of how `iostat.ksh` is used to monitor activity:

```
$ iostat.ksh -c "FIRST EXECUTION" -o 26 -s 1
[Created: /home/tetsur3/results/26/bak/iostat_11.out.39538]
[Created: /home/tetsur3/results/26/bak/iostat_110.sum.39538]
[Creating: /home/tetsur3/results/26/iostat_110.sum]
FIRST EXECUTION
----------------------------------------
Invocation: iostat.ksh -c FIRST EXECUTION -o 26 -s 1
Fri Jun 2 02:51:03 PDT 2000

Hit Ctrl-C to stop...
[Creating: /home/tetsur3/results/26/iostat_11.out]
Waiting for disk i/o ...
-IOWAIT----hdisk0---
W     1.8|      124
-IOWAIT----hdisk0----hdisk1----hdisk4---
W     2.8|       24|       12|       32
R    13.2|         |         |     2625
R    11.5|         |         |     3712
W    11.5|      100|         |
R     3.8|         |         |     4152
R     3.0|        8|         |     1552
W     3.0|      116|         |
W     0.5|       40|         |
W        |        8|         |
```

The source file and the complete syntax of this sample script are included in Appendix A, “Sample scripts” on page 335.

5.3.4 List paging space — lsps

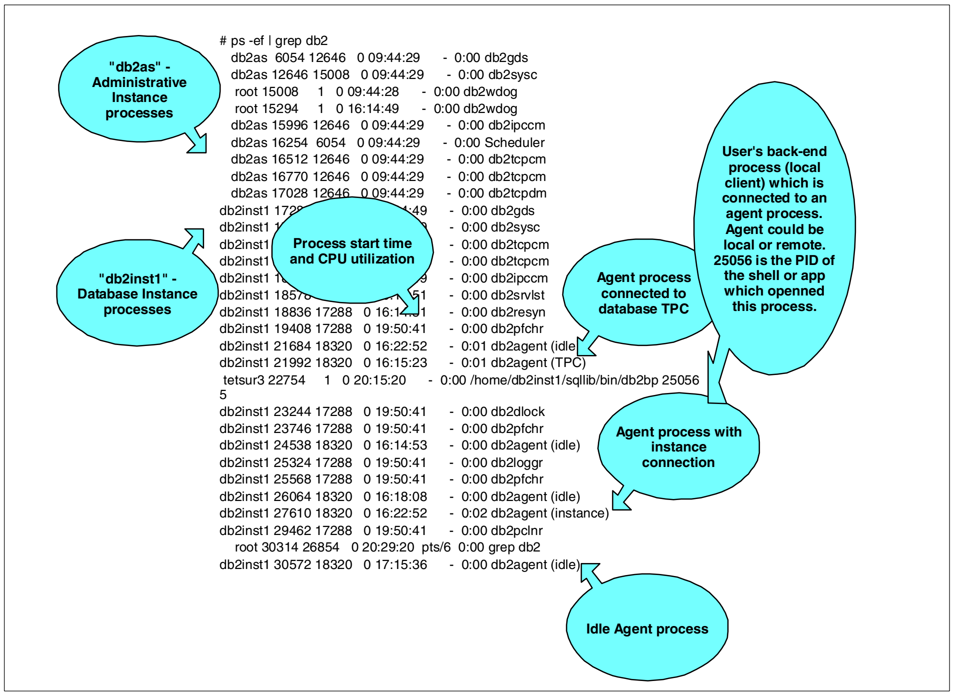
To check paging activity of the operating system, use the AIX command `lsps -a`. This command shows how much paging space has been defined, and how much is being used at the moment, as follows:

```
# lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto  Type
hd6         hdisk0            rootvg        1024MB       1     yes   yes    lv
```

Another way to see this information is to use the `nmon` utility (see 5.3.1, “Online monitor — nmon” on page 128).

5.3.5 Process state — ps

To check which processes of DB2 are running, use the AIX command `ps`. Use `ps -ef | grep db2` to see all of the process. Or use `ps -ef | grep db2agent` to see agents which are idle, handling a database connection or an instance connection. Figure 28 shows a sample invocation of the `ps` command and an explanation of the processes.



```
# ps -ef | grep db2
   db2as  6054 12646   0 09:44:29     -  0:00 db2gds
   db2as 12646 15008   0 09:44:29     -  0:00 db2sysc
    root 15008     1   0 09:44:28     -  0:00 db2wdog
    root 15294     1   0 16:14:49     -  0:00 db2wdog
   db2as 15996 12646   0 09:44:29     -  0:00 db2ipccm
   db2as 16254  6054   0 09:44:29     -  0:00 Scheduler
   db2as 16512 12646   0 09:44:29     -  0:00 db2tcpcm
   db2as 16770 12646   0 09:44:29     -  0:00 db2tcpcm
   db2as 17028 12646   0 09:44:29     -  0:00 db2tcpdm
db2inst1 172          :49     -  0:00 db2gds
db2inst1 1                    -  0:00 db2sysc
db2inst1                      -  0:00 db2tcpcm
db2inst1                      -  0:00 db2tcpcm
db2inst1 1                    -  0:00 db2ipccm
db2inst1 1857           51    -  0:00 db2srvlst
db2inst1 18836 17288   0 16:1   1     -  0:00 db2resyn
db2inst1 19408 17288   0 19:50:41     -  0:00 db2pfchr
db2inst1 21684 18320   0 16:22:52     -  0:01 db2agent (idle
db2inst1 21992 18320   0 16:15:23     -  0:01 db2agent (TPC)
 tetsur3 22754     1   0 20:15:20     -  0:00 /home/db2inst1/sqllib/bin/db2bp 25056
5
db2inst1 23244 17288   0 19:50:41     -  0:00 db2dlock
db2inst1 23746 17288   0 19:50:41     -  0:00 db2pfchr
db2inst1 24538 18320   0 16:14:53     -  0:00 db2agent (idle)
db2inst1 25324 17288   0 19:50:41     -  0:00 db2loggr
db2inst1 25568 17288   0 19:50:41     -  0:00 db2pfchr
db2inst1 26064 18320   0 16:18:08     -  0:00 db2agent (idle)
db2inst1 27610 18320   0 16:22:52     -  0:02 db2agent (instance)
db2inst1 29462 17288   0 19:50:41     -  0:00 db2pclnr
    root 30314 26854   0 20:29:20 pts/6  0:00 grep db2
db2inst1 30572 18320   0 17:15:36     -  0:00 db2agent (idle)
```

"db2as" - Administrative Instance processes
"db2inst1" - Database Instance processes
Process start time and CPU utilization
Agent process connected to database TPC
User's back-end process (local client) which is connected to an agent process. Agent could be local or remote. 25056 is the PID of the shell or app which openned this process.
Agent process with instance connection
Idle Agent process

Figure 28. DB2 processes

5.4 DB2 UDB tools

DB2 UDB provides several tools that can be used for monitoring or analyzing your database. In this section we discuss these monitoring and analyzing tools, which are used for the following purposes:

- **Snapshot Monitor**
  Capturing performance information at periodic points of time.
- **Event Monitor**
  Providing a summary of activity at the completion of events such as statement execution, transaction completion, or when an application disconnects.
- **Explain Facility**
  Providing information about how DB2 will access the data in order to resolve the SQL statements.
- **db2batch tool**
  Providing performance information (benchmarking tool).

- **CLI/ODBC/JDBC Trace Facility**
  Tracing all the function calls of DB2 CLI Driver, for problem determination and tuning applications using CLI, ODBC, or SQLJ, or just to better understand what a third-party application is doing.

The following are some guidelines to determine which tool you should use:

- Choose the Snapshot Monitor or Event Monitor if you want to gather data about DB2's operation, performance, and the applications using it. This data is maintained as DB2 runs and can provide important performance and troubleshooting information.
- Choose the Explain Facility if you want to analyze the access plan for an SQL statement or a group of SQL statements.
- Choose the db2batch tool if you want to measure and analyze the performance of a set of SQL statements. Performance times can be returned along with Snapshot data for analysis. Explain information can be gathered for use by the Explain Facility.
- Choose the CLI/ODBC/JDBC Trace Facility to track activity between a CLI client and DB2. This facility can help pinpoint long running statements and analyze the time spent in the client application, DB2, or the network.

Some of the monitoring tools include information collected by one or more of the other monitoring tools. For example, db2batch and the Event Monitor also display information collected by the Snapshot Monitor.

Note

`MON_HEAP_SZ` indicates the amount of memory (in 4K pages) which is allocated for database monitor data (at db2start). The amount of memory needed will depend on the number of snapshot switches which are turned on and active Event Monitors. If the memory heap is insufficient, an error will be returned when trying to activate a monitor and it will be logged to the `db2diag.log` file.

5.4.1 Obtaining database access information

The first step in the database monitoring process is defining your objectives. Defining the objectives is very important in selecting the best facility to meet your requirements. An objective can be:

- Understanding how a given query will be optimized in a specific environment. For example, there is a query used in an application that does not perform well.

- Understanding how applications use database manager resources at a specific point of time. For example, database concurrency is reduced if a specific application is started.
- Understanding which database manager events have occurred when running applications. For example, you notice a degradation in overall performance when certain applications are run.

5.4.2 Snapshot monitor

The Snapshot Monitor collects various levels of information on the database manager or database objects. The information is always maintained as a point-in-time value, such as a counter, high water mark, or last timestamp for a particular object. No history information is maintained. The different levels of information are:

- Database manager
- Databases (local, remote or DCS)
- Applications (local, remote or DCS)
- FCM (for internal communications between db2 agents)
- Buffer pools
- Table spaces
- Tables
- Locks
- Dynamic SQL statements

While there is always some basic information which is collected for each of these levels by default, it is possible to turn on the collection of a broader range of data for each level by turning on a Snapshot Monitor switch. The switches are defined by the groups listed in Table 10 . The switches can be set at the instance level using the `UPDATE DBM CFG` command or the application level using `UPDATE MONITOR SWITCHES`. Snapshot Monitor switches and the monitoring levels are combined to provide different monitoring information when taking a snapshot. The two are very closely related. If the proper monitor switch is not turned on, the snapshot level used may not return any data.

Note

If you turn on the monitor switch UOW, only the next UOW will be monitored.

Table 10. Snapshot monitor switch groups

Group	Information provided	Monitor switch	DBM CFG parameter
Sorts	Number of heaps used, overflows, sorts performance	SORT	DFT_MON_SORT
Locks	Number of locks held, number of deadlocks	LOCK	DFT_MON_LOCK
Tables	Measure activity (rows read, rows written)	TABLE	DFT_MON_TABLE
Buffer pools	Number of reads and writes, time taken	BUFFERPOOL	DFT_MON_BUFPOOL
Unit of work	Start times, end times, completion status	UOW	DFT_MON_UOW
SQL statements	Start time, stop time, statement identification	STATEMENT	DFT_MON_STMT

5.4.2.1 Scope of snapshot monitor data

When activating a switch from the application level, for example, by issuing the `UPDATE MONITOR SWITCHES` command from the CLP command line, an instance connection is made. All data collected for the switch group is made available to it until the connection is terminated (for example, with the terminate command or by exiting the current shell). The data collected (and displayed) will be different (unique) from any other user or shell which turns on the same switches with the same command at a different point in time.

Figure 29 shows that once the monitor switch was turned on in Application A, the “get Snapshot Monitor” command can report data collected from the activity performed in Application B. Even if the switch is turned on from Application B later, it will be too late to display the activity which has already passed. It is, however, possible for Application B to display data from subsequent activity. So it is clear that different shell/sessions (instance connections) maintain their own set of snapshot data, allowing each to display different information based on active switches, when they were turned on, and whether snapshot data was reset.

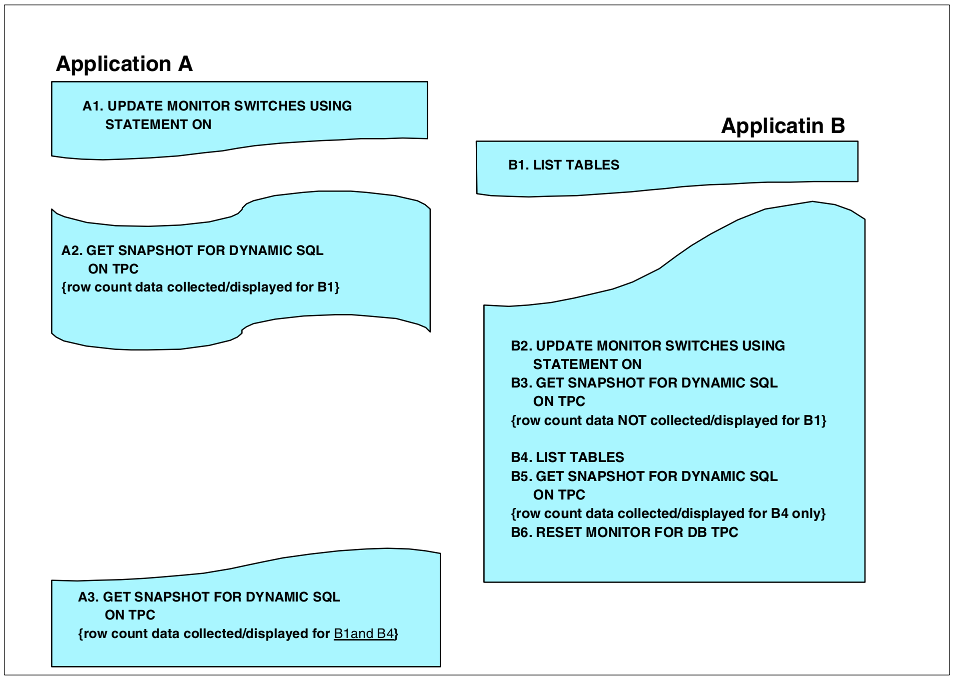

Application A
A1. UPDATE MONITOR SWITCHES USING STATEMENT ON
A2. GET SNAPSHOT FOR DYNAMIC SQL ON TPC
{row count data collected/displayed for B1}
A3. GET SNAPSHOT FOR DYNAMIC SQL ON TPC
{row count data collected/displayed for B1and B4}
Applicatin B
B1. LIST TABLES
B2. UPDATE MONITOR SWITCHES USING STATEMENT ON
B3. GET SNAPSHOT FOR DYNAMIC SQL ON TPC
{row count data NOT collected/displayed for B1}
B4. LIST TABLES
B5. GET SNAPSHOT FOR DYNAMIC SQL ON TPC
{row count data collected/displayed for B4 only}
B6. RESET MONITOR FOR DB TPC

Figure 29. Snapshot data not shared between sessions

In order to make the snapshot information available and consistent for all instance connections, the default monitor switches should be turned on from the Database Configuration parameters. For example:

```
UPDATE DBM CFG USING DFT_MON_STMT ON
```

See Table 10 on page 137 for a list of database manager configuration parameters which determine the default values of the monitor switches.

Note

When you change the value of the database manager configuration parameters, you usually need to stop and start the instance to make those changes effective; however, the changes of the default monitor switches will be effective immediately. Therefore, you do not need to stop and start the instance.

5.4.2.2 Reviewing the Snapshot Monitor switch status

At any time, you can determine the current settings of database monitor switches by issuing the following command:

```
GET MONITOR SWITCHES
```

The following example shows the switch states. The timestamps correspond to the last time the switches were reset or turned on.

```
 Monitor Recording Switches

Switch list for node 0
Buffer Pool Activity Information  (BUFFERPOOL) = ON
Lock Information                        (LOCK) = OFF
Sorting Information                     (SORT) = OFF
SQL Statement Information          (STATEMENT) = ON  06-30-2000 15:20:58.574608
Table Activity Information             (TABLE) = ON  06-30-2000 15:53:23.443439
Unit of Work Information                 (UOW) = OFF
```

The GET `MONITOR SWITCHES` command outputs the status of the monitor switches for the current session. If you issue `GET MONITOR SWITCHES` from the other session, the output may be different because each application using the database system monitor interface has its own set of monitor switches. If you want to see database manager-level switches status, execute the following command:

```
GET DBM MONITOR SWITCHES
```

This command is used to determine if the database system monitor is currently collecting data for any monitoring application. The output will be like the following:

```
 DBM System Monitor Information Collected

Switch list for node 0
Buffer Pool Activity Information  (BUFFERPOOL) = ON  06-30-2000 15:53:30.122255
Lock Information                        (LOCK) = OFF
Sorting Information                     (SORT) = OFF
SQL Statement Information          (STATEMENT) = ON  06-30-2000 15:20:58.574608
Table Activity Information             (TABLE) = ON  06-30-2000 15:53:23.443439
Unit of Work Information                 (UOW) = OFF
```

You can see that the buffer pool monitor switch is on in this sample output. This means that there is another session which turned the buffer pool monitor switch on.

5.4.2.3 Resetting the Snapshot Monitor switches

Some snapshot data will be reset when all activity ends for the database object level. For example, when all connections to a database are closed, the database level information is cleared. In addition, since the statement cache is cleared when all databases disconnect, that information is also cleared from the Snapshot Monitor.

Note

If you want to make sure that snapshot data (as well as the statement cache) remains populated even after the last connection is closed, then be sure to execute `ACTIVATE DATABASE` to keep the database activated. In this case, even if there are no connections, the monitored and cached data is preserved.

Monitor switches can be reset at any time by issuing the command:

```
RESET MONITOR FOR DATABASE databasename
```

Resetting the monitor switches effectively starts all of the counters at zero, and further snapshots are based on the new counter values.

To reset the monitor switches for all databases within an instance, the `RESET MONITOR ALL` command should be used.

Note

Every application has its own copy of the Snapshot Monitor values. Resetting the monitor switches only affects the counters of the application that issues the reset.

5.4.2.4 Retrieving and displaying Snapshot Monitor data

The Snapshot Monitor data is retrieved by entering the `GET SNAPSHOT` command (see the *Administrative API Reference*, SC09-2947 for API equivalents). You can specify the database or application which you want to monitor within the `GET SNAPSHOT` command.

The following example shows an example of snapshot data returned for the Dynamic SQL Statement cache. Note that you need to set the STATEMENT monitor switch ON to monitor all elements shown in this example.

```
$ db2 get snapshot for dynamic sql on SAMPLE

           Dynamic SQL Snapshot Result

 Database name                      = TPC

 Database path                      = /database/db2inst1/NODE0000/SQL00001/

 Number of executions               = 1
 Number of compilations             = 1
 Worst preparation time (ms)        = 13
 Best preparation time (ms)         = 13
 Rows deleted                       = 0
 Rows inserted                      = 0
 Rows read                          = 17
 Rows updated                       = 0
 Rows written                       = 0
 Statement sorts                    = 0
 Total execution time (sec.ms)      = 0.020399
 Total user cpu time (sec.ms)       = 0.010000
 Total system cpu time (sec.ms)     = 0.010000
 Statement text                     = SELECT NAME, CREATOR, TYPE, CTIME FROM
SYSIBM.SYSTABLES WHERE CREATOR = USER ORDER BY CREATOR, NAME
```

As you can see in this example, you can use this information to know currently cached dynamic SQL statements and their statistics.

You can also get data related to the acquired locks using the application level or database level snapshot using the command below. Partial output follows:

```
$ db2 get snapshot for locks on tpc

            Database Lock Snapshot

Database name                              = TPC
Database path                              = /database/db2inst1/NODE0000/SQL0000
1/
Input database alias                       = TPC
Locks held                                 = 6
Applications currently connected           = 2
Agents currently waiting on locks          = 0
Snapshot timestamp                         = 06-30-2000 16:53:57.984177

Application handle                         = 36
Application ID                             = *LOCAL.db2inst1.000630235242
Sequence number                            = 0001
Application name                           = db2bp
Authorization ID                           = DB2INST1
Application status                         = UOW Waiting
Status change time                         = Not Collected
Application code page                      = 819
Locks held                                 = 5
Total wait time (ms)                       = Not Collected
List Of Locks
 Lock Object Name            = 65795
 Node number lock is held at = 0
 Object Type                 = Row
 Tablespace Name             = TPCDDATA
 Table Schema                = DB2INST1
 Table Name                  = REGION
 Mode                        = X
 Status                      = Granted
 Lock Escalation             = NO
.............
```

By analyzing the output, you can see which application holds which type of locks on which database objects.

See the manual *System Monitor Guide and Reference*, SC09-2956 for a detailed explanation of each output element. For the complete syntax of the GET SNAPSHOT command, see the *SQL Reference*, SC09-2951.

5.4.2.5 Sample shell scripts

We have written sample shell scripts to filter and format the output of the dynamic SQL snapshot to make it more readable. One is for the dynamic SQL snapshot, the other one is for the database lock snapshot.

Filter and format the dynamic SQL snapshot

The sample script `sqlcache.ksh` displays the SQL statements and selected statistics currently in the dynamic SQL cache. Some of the statistics will only be displayed if the STATEMENTS monitor switch has been turned on. The columns displayed can also be determined by the parameters to the utility.

The sample invocation shown below extracts all SQL statements from the dynamic SQL cache of the TPC database. The specified comment is added to the output which is saved into `/home/tetsur3/work/results/26` directory. The original output of the dynamic SQL snapshot is also saved into the same directory. The -s option is used to specify the string length of the reported SQL statements.

```
$ sqlcache.ksh -c "Just SQL Stmts" -d tpc -o 26 -s 80 -sql -b
[Creating: /home/tetsur3/results/26/sqlcache_TPC_800100.sum]
-- Just SQL Stmts
-- ---------------------------------------
-- Invocation: /home/tetsur3/work/sqlcache.ksh -c Just SQL Stmts -d tpc -o 26 -s
 80 -sql -b
-- Fri Jun 2 17:44:24 PDT 2000
--
db2 get snapshot for dynamic sql on TPC

[Creating: /home/tetsur3/results/26/sqlcache_TPC.out]

SQL-Text
--------------------------------------------------------------------------------
SELECT NAME, CREATOR, TYPE, CTIME FROM SYSIBM.SYSTABLES WHERE CREATOR = USER ORD
select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comme
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedpri
select count(*) from customer
SELECT colcount FROM SYSIBM.SYSTABLES WHERE creator = 'SYSIBM' AND name= 'SYSTAB
```

The following sample shows sample invocations to extract all SQL statements from the dynamic SQL cache of the TPC database with statistics. The following sample shows partial statistics of extracted SQL statements:

```
$ sqlcache.ksh -c "Initial Cache Contents" -d tpc -o 26 -s 3>
[Creating: /home/tetsur3/results/26/sqlcache_TPC_331011.sum]
-- Initial Cache Contents
-- ---------------------------------------
-- Invocation: /home/tetsur3/work/sqlcache.ksh -c Initial Cache Contents -d tpc
-o 26 -s 33 -f -t -w -b
-- Fri Jun 2 17:37:17 PDT 2000
--
db2 get snapshot for dynamic sql on TPC

[Creating: /home/tetsur3/results/26/sqlcache_TPC.out]

Qnum | Exec's Comp's | BestPrepMS | SQL-File | SQL-Text
-------------------------------------------------------------------------------
|  1 |      1     1 |         10 |   23.sql | SELECT NAME, CREATOR, TYPE, CTIME
|  2 |      2     1 |        108 |   27.sql | select s_acctbal, s_name, n_name,
|  3 |      1     1 |         27 |   28.sql | select l_returnflag, l_linestatus
|  4 |      1     1 |          5 |  155.sql | select count(*) from customer
|  5 |      3     1 |         19 |   26.sql | SELECT colcount FROM SYSIBM.SYSTA
```

The following sample shows all statistics of extracted SQL statements:

```
$ sqlcache.ksh -c "AFTER NUM_IO CHANGES" -d tpc -o 26 -s 130
[Created: /home/tetsur3/results/26/bak/sqlcache_TPC.out.27994]
[Created: /home/tetsur3/results/26/bak/sqlcache_TPC_1300000.sum.27994]
[Creating: /home/tetsur3/results/26/sqlcache_TPC_1300000.sum]
-- AFTER NUM_IO CHANGES
-- ---------------------------------------
-- Invocation: /home/tetsur3/work/sqlcache.ksh -c AFTER NUM_IO CHANGES -d tpc -o 26 -s 1
-- Fri Jun 2 17:26:32 PDT 2000
--
db2 get snapshot for dynamic sql on TPC

[Creating: /home/tetsur3/results/26/sqlcache_TPC.out]

Qnum | Exec's Comp's | BestPrepMS | ExecSEC.MS UserSEC.MS SystSEC.MS |
|R_Read Writtn| Inserted  Updated  Deleted|SQL-Text
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------------------------------------------------
-----------------------
|  1 |      1      1 |         10 |   0.012103          0          0 |
|     17      0 |       0          0        0 | SELECT NAME, CREATOR, TYPE,
 CTIME FROM SYSIBM.SYSTABLES WHERE CREATOR = USER ORDER BY CREATOR, NAME
|  2 |      2      1 |        108 |    2.15153      1.435       0.21 |
|   4648    460 |       0          0        0 | select s_acctbal, s_name,
n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment from part, supplier, p
artsupp, nation, region w
|  3 |      1      1 |         27 |    110.448     109.69       0.56 |
|     6M      0 |       0          0        0 | select l_returnflag, l_line
status, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice * (1 -
|  4 |      1      1 |          5 |   0.688492       0.58       0.02 |
|      4      0 |       0          0        0 | select count(*) from customer
|  5 |      3      1 |         19 |          0          0          0 |
|      0      0 |       0          0        0 | SELECT colcount FROM SYSIBM
.SYSTABLES WHERE creator = 'SYSIBM' AND name='SYSTABLES'
```

The source file and the complete syntax of this sample script are included in Appendix A, “Sample scripts” on page 335.

Filter and format the database locks snapshot

The sample script `locks.ksh` displays the current locks held in the database as reported by the locks Snapshot Monitor. All of the information displayed is collected even without turning the `LOCKS` monitor switch `ON`. The number of locks are displayed at the database, application and table levels. The database and application level information can be discarded by specifying parameters. See the following example:

```
$ locks.ksh -c "Region Locks" -d tpc -o 26 -b
[Creating: /home/tetsur3/results/26/locks_TPC_0.sum]
-- Region Locks
-- ---------------------------------------
-- Invocation: /home/tetsur3/work/locks.ksh -c Region Locks -d tpc -o 26 -b
-- Fri Jun 2 17:57:00 PDT 2000
--
db2 get snapshot for locks on TPC

[Creating: /home/tetsur3/results/26/locks_TPC.out]


Database name                           = TPC
Database path                           = /database/db2inst1/NODE0000/SQL00001/
Input database alias                    = TPC
Locks held                              = 4
Applications currently connected        = 1
Agents currently waiting on locks       = 0
Snapshot timestamp                      = 06-02-2000 17:57:01.864817

APP.NAME   APP.USER HANDLE APP.ID              APP.STATUS      LOCKS WAIT.ms
---------- -------- ------ ------------------- --------------- ----- -------
db2bp      DB2INST1      2 *LOCAL.db2inst1      UOW Waiting         4

TABLE NAME                     | TYPE       | ESC | MODE | STATUS     | COUNT
------------------------------ | ---------- | --- | ---- | ---------- | -----
=.=                            | Lock       | NO  | S    | Granted    |     1
DB2INST1.REGION                | Row        | NO  | X    | Granted    |     2
DB2INST1.REGION                | Table      | NO  | IX   | Granted    |     1
```

The `COUNT` column indicates the number of locks held for the table object with the same Type, Escalation (YES or NO), Mode and Status.

Note

The "table" with the name "=.=" indicates an internal temporary table. See the *Administration Guide: Implementation*,SC09-2944 for further details on TYPE and MODE of locks.

5.4.3 Event Monitor

While Snapshot Monitoring records the state of database activity when the snapshot is taken, an Event Monitor records the database activity every time an *event* or *transition* occurs. Some database activities that need to be monitored cannot be easily captured using the Snapshot Monitor. These activities include deadlock scenarios. When a deadlock occurs, DB2 will resolve the deadlock by issuing a ROLLBACK for one of the transactions. Information regarding the deadlock event cannot be easily captured using the Snapshot Monitor, since the deadlock has probably been resolved before a snapshot can be taken.

Event Monitors are created using SQL DDL (Data Definition Language) like other database objects. Event Monitors can be turned on or off much like the Snapshot Monitor switches.

Note

SYSADM or DBADM authority is required to create an Event Monitor.

When an Event Monitor is created, the type of event to be monitored must be stated. The Event Monitor can monitor the following events:

- **DATABASE** — Records an event record when the last application disconnects from the database.
- **TABLES** — Records an event record for each active table when the last application disconnects from the database. An active table is a table that has changed since the first connection to the database.
- **DEADLOCKS** — Records an event record for each deadlock event.
- **TABLESPACES** — Records an event record for each active table space when the last application disconnects from the database.
- **BUFFERPOOLS** — Records an event record for buffer pools when the last application disconnects from the database.
- **CONNECTIONS** — Records an event record for each database connection event when an application disconnects from a database.
- **STATEMENTS** — Records an event record for every SQL statement issued by an application (dynamic and static).
- **TRANSACTIONS** — Records an event record for every transaction when it completes (COMMIT or ROLLBACK statement).

As shown above, many types of Event Monitors generate an event record when the last application is disconnected. However, you can use the `FLUSH EVENT MONITOR` command to write out current database monitor values for all active monitor types associated with a particular Event Monitor.

```
db2 FLUSH EVENT MONITOR evmon_name
```

The event records written out by this command are noted in the Event Monitor log with *a partial record identifier*. You should be aware that flushing out the Event Monitor will not cause the Event Monitor values to be reset. This means that the Event Monitor record that would have been generated if no flush was performed will still be generated when the normal monitored event is triggered.

Note

Event Monitors can be created using either using SQL or the db2emcrt GUI tool.

The output of an Event Monitor is stored in a directory or in a named pipe. The existence of the pipe or the file will be verified when the Event Monitor is activated. If the target location for an Event Monitor is a named pipe, then it is the responsibility of the application to read the data promptly from the pipe.

Note

If you're interested in only filtering certain information from the Event Monitor, using a pipe is recommended in order to save disk space (and maintenance) for the monitor storage files. See 5.4.3.6, "Working with Event Monitors using a pipe" on page 155 for an example of using an Event Monitor with a pipe.

If the target for an Event Monitor is a directory, then the stream of data will be written to a series of files. The files are sequentially numbered and have a file extension of `evt` (such as 00000000.evt, 00000001.evt, and so on). The maximum size and number of Event Monitor files is specified when the monitor is defined.

Note

An Event Monitor will turn itself off if the defined file space has been exceeded. A message is written to db2diag.log and db2err.log.

5.4.3.1 Creating an Event Monitor

As mentioned, Event Monitors are database objects created using `CREATE EVENT MONITOR` statements. The `db2emcrt` GUI tool can also be used to create Event Monitors.

As an example, you could use the following DDL to create an Event Monitor to track transactions and statements:

```
DROP   EVENT MONITOR mon_tr_st;
CREATE EVENT MONITOR mon_tr_st FOR
    TRANSACTIONS,
    STATEMENTS
  WRITE TO FILE
     -- don't forget to create directory & chmod g+rw
    '/eventmonitors/mon_tr_st'
     -- each file 2 Mb (4k pages)
    MAXFILESIZE 500
     -- 10 files, so maximum 20 Mb
    MAXFILES 10
     -- 64 K buffer, large buffer for very active monitor
    BUFFERSIZE 16
     -- Replace files each time monitor started
    REPLACE
     -- monitor not started automatically at db2start
    MANUALSTART;
```

This Event Monitor is defined to allocate up to ten files, each 2 MB in size, for a total monitor storage area of 20 MB. Other Event Monitor options include specifying the size of the write buffer, synchronous (blocked) writes, asynchronous (unblocked) writes, APPEND the Event Monitor data to existing records, or REPLACE the Event Monitor data in the directory whenever the monitor is activated.

Event Monitors can be defined to monitor many different types of database activities. A filter can also be specified for an Event Monitor. The filter can be based on the application ID, authorization ID or application name, such as AUTH_ID = 'DB2INST1', APPL_NAME = 'db2batch').

Note

The Event Monitor output directory will not be created by DB2. It must be created by the database administrator, and the instance owner must be able to write to the specified directory.

5.4.3.2 Event Monitor states

After creating the Event Monitor, and the specified directory, you must activate the monitor to begin collecting data (unless the monitor was defined with AUTOSTART option). You could use the following DB2 command, specifying 1 to activate or 0 to deactivate the monitor.

```
db2 SET EVENT MONITOR mon_tr_st STATE 1
```

Note

The GUI tool db2emcrt allows you to create and start an Event Monitor in the one operation. It also allows you to stop the monitor.

To check whether an Event Monitor is currently enabled or disabled, you can use the following SQL statement:

```
SELECT  evmonname
        , EVENT_MON_STATE(evmonname)    AS state
        , io_mode
FROM    syscat.eventmonitors;
```

The output would be as follows:

```
EVMONNAME          STATE       IO_MODE
------------------ ----------- -------
MON_TR_ST                    0 B
MON_ALL_PIPE                 0 -
```

The IO_MODE column indicates if the monitor uses a blocked file. The column will be null for monitors using a names pipe.

To see which events were defined for each Event Monitor, you could use the following statements:

```
SELECT
     EVMONNAME
   , TYPE
   , VARCHAR( FILTER, 40) AS FILTER40
FROM
   SYSCAT.EVENTMONITORS
;
```

The output would be as follows:

```
EVMONNAME          TYPE               FILTER40
------------------ ------------------ ---------------------
MON_TR_ST          TRANSACTIONS       -
MON_TR_ST          STATEMENTS         -
MON_ALL_PIPE       DATABASE           -
MON_ALL_PIPE       TABLES             -
MON_ALL_PIPE       DEADLOCKS          -
MON_ALL_PIPE       TABLESPACES        -
MON_ALL_PIPE       BUFFERPOOLS        -
MON_ALL_PIPE       CONNECTIONS        -
MON_ALL_PIPE       TRANSACTIONS       -
MON_ALL_PIPE       STATEMENTS         -

  10 record(s) selected.
```

Tip

Note that this example uses the VARCHAR function to truncate the FILTER column rather that the SUBSTR function. In this way you will receive a warning (SQL0445W) if the columns had values which were truncated.

Note

There is no limit in the number of defined Event Monitors, but a maximum of 32 Event Monitors can be active per DB2 instance at a time.

5.4.3.3 Removing an Event Monitor

Just like other database objects, Event Monitors can be dropped from the database. Removing the definition will remove the associated rows in the system catalog views, `SYSCAT.EVENTMONITORS` and `SYSCAT.EVENTS`. An example of removing the `evmon1` Event Monitor is as follows:

```
DROP EVENT MONITOR evmon1
```

5.4.3.4 Event Monitor records

Event Monitor files cannot be analyzed directly. An application must be used. There are a few alternatives provided by DB2 for analyzing Event Monitor data that we will discuss, but let us first examine some of the Event Monitor records.

To ensure that all of the event records have been written to disk (some may be buffered), you could simply turn the Event Monitor off. You can also use the `BUFFER` option of `FLUSH EVENT MONITOR` command as follows:

```
FLUSH EVENT MONITOR evmon1 BUFFER
```

This forces the Event Monitor buffers to be written out. The `FLUSH EVENT MONITOR` command with the `BUFFER` option does not generate a partial record. Only the data already present in the Event Monitor buffers are written out.

Event monitoring is similar to tracing, since each event is recorded as it occurs, and it is appended to the event record log files (although you may not see it right away because of Event Monitor buffering). An Event Monitor file will contain a number of event records covering event types including:

- Connections
- SQL statements
- Transactions
- Deadlocks
- Buffer pool events
- Table space events
- Table events

If an Event Monitor is monitoring database, table space, or table events, it will write complete event records when the last application using the database disconnects. As already discussed, you can use `FLUSH EVENT MONITOR` command to get the partial event records.

Analyzing Event Monitor output

You can use the db2evmon utility to generate a report to analyze Event Monitor data.

To generate the Event Monitor report for the `mon_tr_st` monitor, issue the following command indicating where the Event Monitor files are located:

```
db2evmon -path /eventmonitors/mon_tr_st
```

or

```
db2evmon -db tpc -evm mon_tr_st
```

The -path option of the db2evmon command is used to indicate the path where the Event Monitor files reside.

The output of the db2evmon utility will be displayed on the screen by default. It is best to redirect the output to a file for analysis.

The following example shows a monitored event record for an SQL statement issued by an applications. You can see executed SQL statements and the statistics for the statements from the monitored event records.

```
.....
12) Statement Event ...
  Appl Handle: 45
  Appl Id: *LOCAL.db2inst1.000519225726
  Appl Seq number: 0001

  Record is the result of a flush: FALSE
  -----------------------------------------
  Type     : Dynamic
  Operation: Close
  Section  : 1
  Creator  : NULLID
  Package  : TOOL1D02
  Cursor   : DYNCUR
  Cursor was blocking: FALSE
  Text     : select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
sum(l_extendedprice) as sum_base_price, sum(l_extendedprice * (1 - l_discount)) as
sum_disc_price, sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as
avg_disc, count(*) as count_order from lineitem where l_shipdate <=
date( '1998-12-01') - 90 day  group by l_returnflag, l_linestatus order by
l_returnflag, l_linestatus
  -----------------------------------------
  Start Time: 05-19-2000 15:57:27.010169
  Stop Time:  05-19-2000 15:59:16.884869
  Exec Time:  109.874700 seconds
  Number of Agents created: 1
  User CPU:   109.080000 seconds
  System CPU: 0.630000 seconds
  Fetch Count: 4
  Sorts: 1
  Total sort time: 60514
  Sort overflows: 0
  Rows read: 6001219
  Rows written: 0
  Internal rows deleted: 0
  Internal rows updated: 0
  Internal rows inserted: 0
  SQLCA:
   sqlcode: 0
   sqlstate: 00000
.....
```

5.4.3.5 Filtering the monitored event record

We have written a sample shell script `mon_stmt.ksh`, which returns information about the SQL statements (For example, `OPEN CURSOR`, `FETCH`) reaching the DB2 Engine as collected by the statements Event Monitor. This sample script issues the `db2evmon` command which reads the event records and generates a report. Before using the script, the Event Monitor has to be created and activated. This script has some parameters to specify what kind of statistics (such as start time, CPU usage, row counts, and sort information) should be extracted.

This script maintains the conventions discussed earlier (see 5.2, "Maintaining tuning information" on page 120) to save output files in the directory structure. The output that the script generates is dependant on the parameters chosen and may include: SQL files for individual statements captured; the entire results of the `db2evmon` command; and the summarized (filtered) information and statistics on each statement.

The following example shows a sample output:

```
-- ----------------------------------------
-- Invocation: mon_stmt.ksh -c PIPE -d tpc -m mon_all -o 24 -s 117
-- Wed May 31 17:56:02 PDT 2000
--
db2evmon TPC mon_all

Hand AppID Seq  Start-Time       | Ty Oper       Code |  Exec(s)  UCPU(s)  SCPU(s) |
   Read  Written Fetch Sorts SOVFL Sort(ms) | SQL-Text
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
  45 25726 0001 15:57:26.520516 | Dy Prepare       0 | 0.038772 0.000000 0.000000 |
         2         0     0     0     0         0 | SELECT colcount FROM SYSIBM.
SYSTABLES WHERE creator = 'SYSIBM' AND name= 'SYSTABLES'
  45 25726 0001 15:57:26.560277 | St Execute       0 | 0.008392 0.000000 0.000000 |
         1         0     0
  45 25726 0001 15:57:26.569503 | St Execute       0 | 0.000682 0.000000 0.000000 |
         1         0     0
  45 25726 0001 15:57:26.582471 | Dy Prepare       0 | 0.423956 0.050000 0.010000 |
        15        37     0     0     0         0 | select l_returnflag, l_linestatus,
sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extended
  45 25726 0001 15:57:27.010169 | Dy Open          0 | 0.000172 0.000000 0.000000 |
         0         0     0     0     0         0 | select l_returnflag, l_linestatus,
sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extended
  45 25726 0001 15:57:27.010169 | Dy Close         0 | 109.8747 109.0800 0.630000 |
   6001219         0     4     1     0     60514 | select l_returnflag, l_linestatus,
sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extended
  45 25726 0001 15:59:16.902719 | St Commit        0 | 0.042587 0.000000 0.000000 |
         0         0     0
```

For the complete syntax and the source file, see Appendix A, “Sample scripts” on page 335.

5.4.3.6 Working with Event Monitors using a pipe

In many cases, using a pipe with an Event Monitor can be very useful. The obvious advantage is in avoiding the maintenance of the growing Event Monitor files. Although you can limit the total size of the Event Monitor files in advance, this may also mean that you might not catch desired events when there is no more room left for the monitor. By using a pipe, you can discard all unwanted data, filtering only on the event information which you want to analyze.

Creating an Event Monitor using a pipe

You could use the following script to create an Event Monitor which captures all events and sends the data to a pipe (this is just an example, and is not necessarily recommended).

```
DROP   EVENT MONITOR mon_all_pipe;
CREATE EVENT MONITOR mon_all_pipe FOR
    DATABASE,
    BUFFERPOOLS,
    TABLESPACES,
    TABLES,
    CONNECTIONS,
    TRANSACTIONS,
    STATEMENTS,
    DEADLOCKS
WRITE TO PIPE
            -- to create: mkfifo -m g+rw /eventmonitors/mon_all_pipe
    '/eventmonitors/mon_all_pipe'
            -- monitor not started automatically at db2start
    MANUALSTART
;
```

Use the AIX command below to create the pipe for the Event Monitor and give it the proper permissions:

```
mkfifo -m g+rw /eventmonitors/mon_all_pipe
```

Piped Event Monitor states

Changing the state of a pipe Event Monitor (active/inactive) is done the same way as for a file Event Monitor, but with the following restriction. You can only activate the pipe Event Monitor (which would begin the flow of event data) after starting some application, such as *db2evmon*, which will read the flow of data from the pipe. Also, when the application reading from the pipe completes (or is halted), the pipe is closed and the Event Monitor is automatically deactivated.

5.4.4 The Explain Facility

If you want to know how a query will be executed by DB2, you must analyze its *access plan*, which is the method for retrieving data from a specific table. The Explain Facility will provide information about how DB2 will access the data in order to resolve the SQL statements.

If you have identified a particular application as the possible source of a performance problem, you need to obtain the SQL statements that the application issues and analyze the access plan for the SQL statements.

The following list summarizes the different ways by which SQL statements can be obtained for analysis:

- Directly from the user or developer, who can extract it from their source code. There may be cases, however, when the SQL statements are generated dynamically from some sort of querying tool, such as Business Objects (R).
- From the Dynamic SQL Snapshot Monitor (also known as the global package cache).
- From the Statements Event Monitor.

Before describing the capabilities and features of the Explain Facility, you need to understand, at a high level, how SQL statements are processed by the DB2 database engine. Each SQL statement is analyzed by DB2; then it is determined how to process the statement during a static bind or when executed dynamically. The method used to retrieve data from tables is called the access plan.

The component within DB2 that determines the access plan to be used is known as the optimizer. During the static preparation of an SQL statement, the SQL compiler is called on to generate an access plan. The access plan contains the data access strategy, including index usage, sort methods, locking semantics, and join methods. The executable form of the SQL statement is stored in the system catalog tables when a `BIND` command is executed (assuming a deferred binding method). This is called a `package`.

Note

The method for retrieving data from a specific table, such as whether indexes are used or not, is called the access path. The access plan involves a set of access paths.

Sometimes, the complete statement is not known at application development time. In this case, the compiler is invoked during program execution to generate an access plan for the query that can be used by the database manager to access the data. Such an SQL statement is called a dynamic SQL statement. The access plans for a dynamic SQL statement are not stored in the system catalogs. They are temporarily stored in memory (known as the global package cache). The compiler will not be invoked if the access plans for the dynamic SQL statements already exist in the package cache.

5.4.4.1 Overview of the SQL compiler

The SQL compiler performs a number of tasks during the creation of the compiled form of the SQL statements. These phases are described below and are also shown in Figure 30. As you can see in this figure, the representation of the query is stored in an internal in-memory structure known as the Query Graph Model.

- **Parse query**

  The first task of the SQL compiler is to analyze the SQL query to validate the syntax. If any syntax errors are detected, the SQL compiler stops processing, and the appropriate SQL error is returned to the application attempting to compile the SQL statement. When parsing is complete, an internal representation of the query is created.

- **Check semantics**

  The second task of the compiler is to further validate the SQL statement by checking to ensure that the parts of the statement make sense given the other parts, for example, ensuring that the data types of the columns input into scalar functions are correct for those functions.
  Also during this stage, the compiler adds the behavioral semantics to the query graph model, such as the effects of referential constraints, table check constraints, triggers, and views.

- **Re-write query**

  The SQL compiler uses global semantics provided in the query graph model to transform the query into a form that can be optimized more easily. For example, the compiler might move a predicate, altering the level at which it is applied, in an attempt to improve query performance. This particular process is called *general predicate pushdown*. Any changes made to the query are re-written back to the query graph model.

- **Optimize access plan**

  The SQL optimizer portion of the SQL compiler uses the query graph model as input and generates many alternative execution plans for satisfying the user's request. It estimates the execution cost of each alternative plan using the statistics for tables, indexes, columns, and functions, and chooses the plan with the smallest estimated execution cost.

The optimizer uses the query graph model to analyze the query semantics and to obtain information about a wide variety of factors, including indexes, base tables, derived tables, sub-queries, correlation, and recursion. The output from this step of the SQL compiler is an access plan, which provides the basis for the information captured in the *Explain tables*. The information used to generate the access plan can be captured with an *Explain snapshot*.

- **Generate executable code**

  The final step of the SQL compiler uses the access plan and the query graph model to create an executable access plan, or section, for the query. This code generation step uses information from the query graph model to avoid repetitive execution of expressions that only need to be computed once for a query. Examples for which this optimization is possible include code page conversions and the use of host variables.

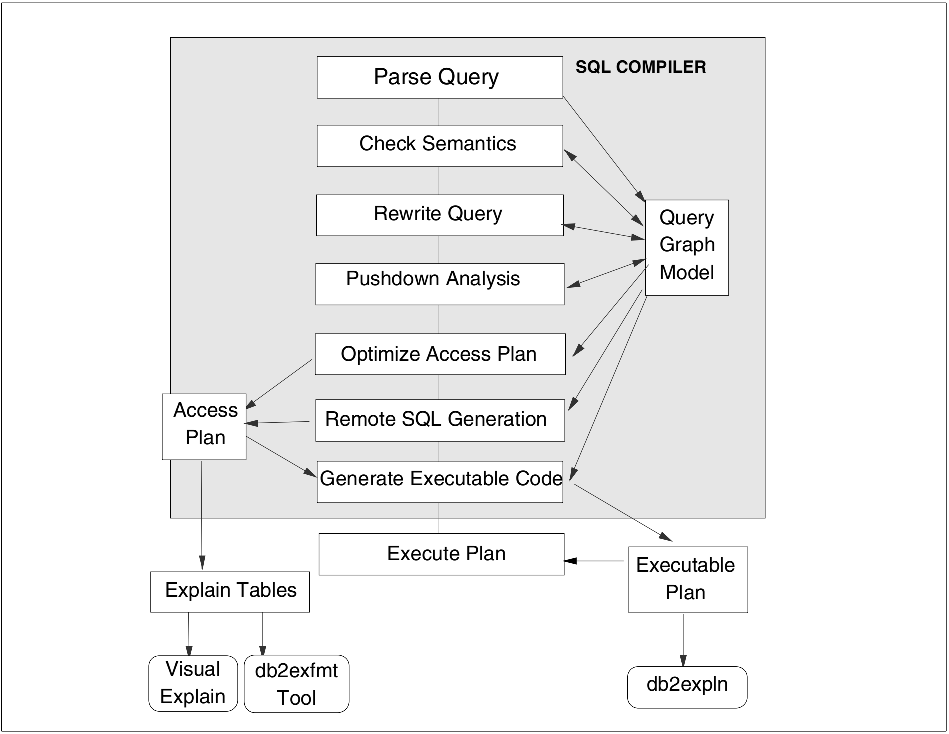


Figure 30. Diagram of steps performed by SQL compiler

Information about access plans for static SQL is stored in the system catalog tables. When the package is executed, DB2 will use the information stored in the system catalog tables to determine how to access the data and provide results for the query. It is this information that is used by the `db2expln` tool.

It is recommended that the `RUNSTATS` command be done periodically on tables used in queries where good performance is desired. The optimizer will then be better equipped with relevant statistical information on the nature of the data. If the `RUNSTATS` command is not run, or the optimizer determines that `RUNSTATS` was run on empty or near empty tables, the optimizer may either use defaults or attempt to derive certain statistics based upon the number of file pages used to store the table on disk.

The Explain information must be captured before you can review it using one of DB2's Explain tools. You can decide to capture detailed information regarding the access plan. While the query is being compiled, the information can be captured into a file or special tables known as Explain tables.

5.4.4.2 Explain tables

DB2 uses Explain tables to store access plan information so that users can see the decisions that the optimizer has made. These tables are called:

- EXPLAIN_ARGUMENT — Represents the unique characteristics for each individual operator.
- EXPLAIN_INSTANCE — Main control table for all Explain information. Each row of data in the Explain tables is explicitly linked to one unique row in this table. Basic information about the source of the SQL statements being Explained and environment information is kept in this table.
- EXPLAIN_OBJECT — Contains data objects required by the access plan generated to satisfy the SQL statement.
- EXPLAIN_OPERATOR — Contains all the operators needed to satisfy the SQL statement.
- EXPLAIN_PREDICATE — Identifies which predicates are applied by a specific operator.
- EXPLAIN_STATEMENT — Contains the text of the SQL statement in two forms. The original version entered by the user is stored in addition to the re-written version that is the result of the compilation process.
- EXPLAIN_STREAM — This table represents the input and output data streams between individual *operators* and *data objects*. Operators involved in a data stream are represented in the EXPLAIN_OPERATOR table. Data objects are represented in the EXPLAIN_OBJECT table.

The Explain tables have to be created before any Explain information can be gathered. The CLP input file, called `EXPLAIN.DDL`, located in the `misc` directory of the SQLLIB directory, contains the definition of the Explain tables. To create the Explain tables, you can connect to the database and use the following command:

```
db2 -tvf EXPLAIN.DDL
```

Note

Explain tables are created the first time you use Visual Explain.

5.4.4.3 Gathering Explain data

There are different kinds of Explain data that can be collected. They differ in Explain table columns that will be populated. The Explain data options are:

- EXPLAIN — Captures detailed information of the access plan and stores the information in the Explain tables. No snapshot information is stored.
- EXPLAIN SNAPSHOT — Captures the current internal representation of an SQL query and related information. The snapshot information is stored in the `SNAPSHOT` column of the `EXPLAIN_STATEMENT` table.

Not all the Explain tools require the same kind of Explain data. Some tools use the data captured using the `EXPLAIN` option and others, such as Visual Explain, require snapshot data.

After creating the Explain tables, you can start capturing the Explain data that will populate them. Not all the SQL statements can be Explained. The Explainable SQL statements include: `SELECT`, `SELECT INTO`, `UPDATE`, `INSERT`, `DELETE`, `VALUES`, and `VALUES INTO` statements.

Depending on the number of SQL statements or kind of application you want to Explain, you should use different methods. These methods include the following:

- EXPLAIN Statement — Gathers Explain data for an SQL statement.
- CURRENT EXPLAIN MODE special register — Specifies to gather Explain data for dynamic SQL statements.
- CURRENT EXPLAIN SNAPSHOT special register — Specifies to gather the Explain snapshot data for dynamic SQL statements.
- BIND Options — Specify to gather Explain data for static and/or dynamic embedded SQL statements in a package.

We will look at each of these methods in turn.

5.4.4.4 EXPLAIN statement

The EXPLAIN statement is useful when you want to gather Explain information for a single dynamic SQL statement. The EXPLAIN statement can be invoked either from the Command Line Processor, Command Center or within an application.

You can control the amount of Explain information that the EXPLAIN statement will store in the Explain tables. The default is to only capture regular Explain table information and not the snapshot information. If you wish to modify this behavior, this is done using the following EXPLAIN statement options:

- **WITH SNAPSHOT**— This option will capture Explain and Explain snapshot data into the Explain tables. This will enable analysis from Visual Explain.
- **FOR SNAPSHOT** — This option only captures the Explain snapshot information. No other Explain information is captured other that normally found in the EXPLAIN_INSTANCE and EXPLAIN_STATEMENT tables.
- The default case is used when no other Explain option is specified. In the default case, the EXPLAIN statement will only gather the Explain data. No Explain snapshot data is captured.

  To issue the EXPLAIN statement, the user must have INSERT privilege on the Explain tables.

Note

The SQL statement being Explained using the EXPLAIN statement will not be executed; only the Explain data is captured.

Let us examine an example of gathering access plan information using the EXPLAIN statement. The Explain statement is shown below. This example collects all the available Explain information for the Explain tables.

```
EXPLAIN ALL WITH SNAPSHOT FOR "SELECT * FROM table1"
```

Note

Instead of the keyword ALL, used in our example, other keywords, PLAN and PLAN SELECTION, can be used. Your EXPLAIN statement must include one of them.

The EXPLAIN statement shown in the last example populates a number of Explain tables, including the SNAPSHOT column of the EXPLAIN_STATEMENT table.

The SNAPSHOT_TAKEN column in the EXPLAIN_INSTANCE table indicates the existence of a Visual Explain snapshot for each Explained statement.

The EXPLAIN statement can also be embedded in an application to populate the Explain tables. Once the Explain tables are populated, they can be queried. Special operators can be examined to determine if the ideal access plan was used for the query.

Explain special register

Another way to collect Explain information is by using the Explain special registers. There are two special registers used by DB2 for gathering Explain information for dynamic SQL statements. These registers can be set interactively, or they can be used in a dynamic embedded SQL program. The values of special registers are modified using the SET statement.

The special registers are:

- CURRENT EXPLAIN MODE — Used to populate only the Explain data. No snapshot will be taken.
- CURRENT EXPLAIN SNAPSHOT — Used to capture only the Explain snapshot data.

The following statements are used to set the value of the Explain special registers:

```
SET CURRENT EXPLAIN MODE     option
SET CURRENT EXPLAIN SNAPSHOT option
```

The Explain registers options are:

- NO — No Explain information is captured for dynamic SQL statements.
- YES — Explain tables or snapshot information will be populated for dynamic SQL statements while executing the SQL statement, and the result is returned.

- EXPLAIN — Explain tables or snapshot information will be populated for dynamic SQL statements without executing the SQL statement. Use this state to obtain Explain information without executing the SQL statement.

The following two options are for the `CURRENT EXPLAIN MODE` register only:

- RECOMMEND INDEXES — This option will be discussed in the section that discusses the Index Advisor.

  EVALUATE INDEXES — This option will be discussed in the section that discusses the Index Advisor.

Note

Once you have set a register to YES or EXPLAIN, any subsequent dynamic SQL statements will be Explained until the register is reset to NO.

Explain BIND options

The `BIND` command prepares SQL statements creates a package that is stored in the database. The BIND command has the options which are related to the Explain information, `EXPLAIN` and `EXPLSNAP`. The `EXPLSNAP` option collects Explain snapshot information. If you want to view the access plan using Visual Explain, then you use the `EXPLSNAP` option. The `EXPLAIN` option only populates the Explain information without including a snapshot.

When `ALL` is specified for the `EXPLAIN` or `EXPLSNAP` options, instead of `YES`, then the Explain tables will also be populated for dynamic SQL.

Note

Explain snapshots cannot be performed for DRDA application servers.

Now let us capture some Explain data using a bind option:

```
BIND program1.bnd EXPLSNAP ALL
```

In our example, the Explain snapshot information will be populated for all of the static SQL statements defined in the `program1.bnd` package. Because the ALL option was specified, the dynamic SQL statements issued during package execution will also have Explain snapshot information gathered at run-time.

The method of obtaining Explain information during binds is useful for an administrator to determine the access plans of static or dynamic statements executed from packages.

To examine the access plan data for individual dynamic SQL statements, the special register technique is an easier method to use.

Using the Explain report tools to gather and analyze Explain data
There are alternative methods of gathering Explain data that is stored in a report rather than in the Explain tables. They are the `dynexpln` tool and the `db2expln` tool.

The `db2expln` tool describes the access plan selected for static SQL statements in the packages stored in the system catalog tables. On the other hand, the `dynexpln` tool describes the access plan selected for dynamic SQL statements. It creates a static package for the statements and then uses the `db2expln` tool to describe them.

The Explain output of both utility programs is stored in a readable report file. The Explain report tools are useful as quick and easy methods for gathering access plan information.

The following example shows the access plan selected for static SQL statement in the package, `CURSOR`. This package has been created by precompiling and binding the sample program `cursor.sqc` which you can find in the `~/sqllib/samples/c` directory.

```
Section = 1

SQL Statement:

  SELECT name, dept
  FROM staff
  WHERE job='Mgr'


Estimated Cost        = 25
Estimated Cardinality = 12

Access Table Name = TETSUYA.STAFF  ID = 2,3
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
|  Sargable Predicate(s)
|  |  #Predicates = 1
|  Return Data to Application
|  |  #Columns = 2
Return Data Completion

End of section
```

By analyzing this output, you can see the information about how DB2 will access the data in order to resolve the SQL statements. In this example, a table scan is chosen to access the STAFF table, and a SARGable predicate is applied (see Chapter 7 for the predicate types).

5.4.4.5 Examining EXPLAIN data

Once the Explain data has being stored in the Explain tables, it can be queried or displayed using Visual Explain or other Explain tools. We will now present how to use Visual Explain to review and analyze an access plan.

Visual Explain

Visual Explain is a GUI (Graphical User Interface) utility that gives the database administrator or application developer the ability to examine the access plan determined by the optimizer. Visual Explain can only be used with access plans Explained using the snapshot option.

Visual Explain can be used to analyze previously generated Explain snapshots or to gather Explain data and Explain dynamic SQL statements. If the Explain tables have not been created when you start Visual Explain, it will create them for you. You can invoke Visual Explain either from the Command Center or Control Center.

From the Control Center interface, right-click on the database where your Explain snapshots are stored. You will notice that there is an option called Show Explained Statements History as shown in Figure 31.

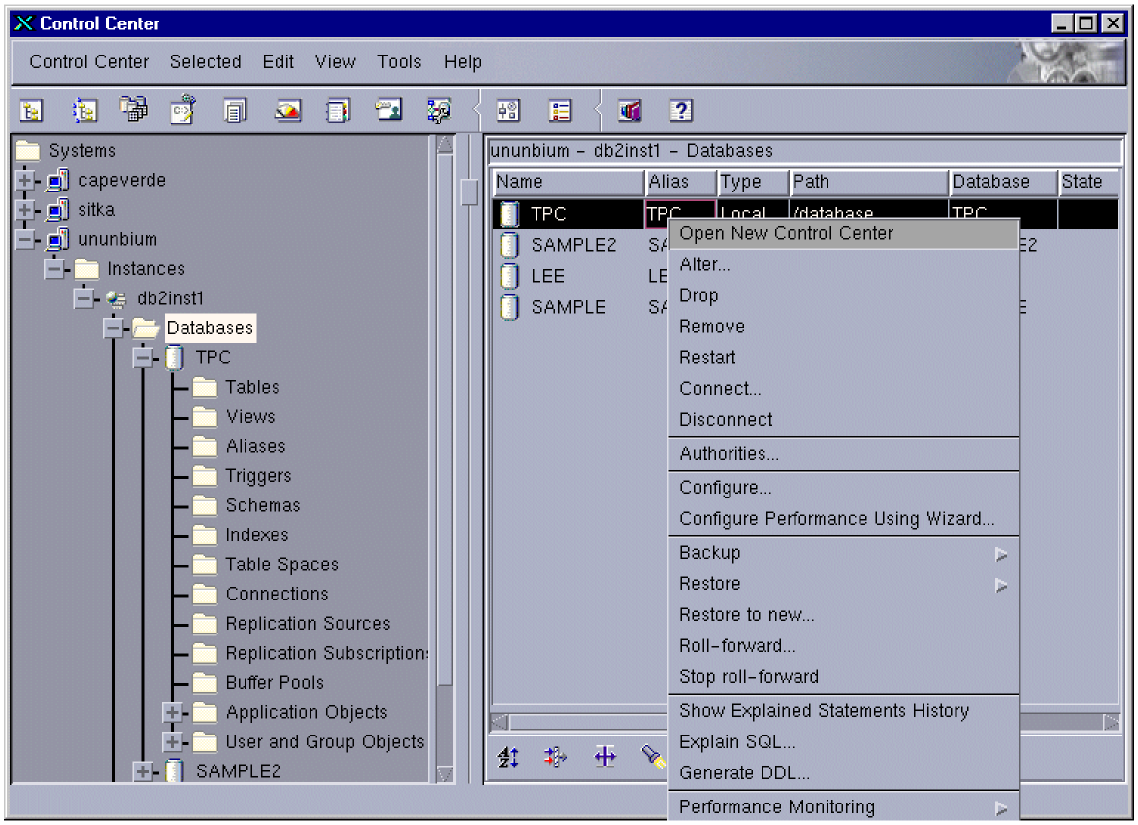
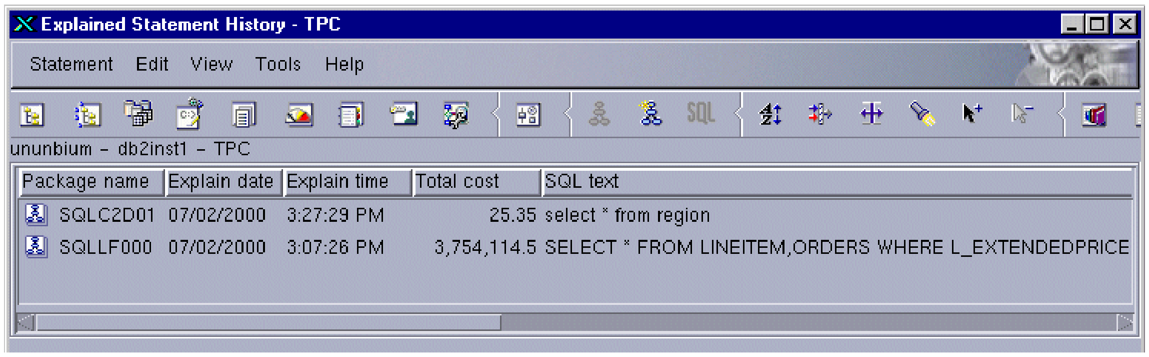


Figure 31. DB2 UDB Control Center — accessing Visual Explain

The **Explain SQL...** option, also shown in Figure 31, allows you to gather Explain data and show the graphical representation of a dynamic SQL statement. This is the easiest way to Explain a single SQL statement.

Once the Explained Statements History window has been opened, all of the Explained statements will be listed as shown in Figure 32. The displayed information may differ since it can be customized to your environment. In Figure 32, the total costs and the SQL statements are shown.


Explained Statement History - TPC
Statement Edit View Tools Help
unbunbium - db2inst1 - TPC

Package name	Explain date	Explain time	Total cost	SQL text
SQLC2D01	07/02/2000	3:27:29 PM	25.35	select * from region
SQLLF000	07/02/2000	3:07:26 PM	3,754,114.5	SELECT * FROM LINEITEM,ORDERS WHERE L_EXTENDEDPRICE

Figure 32. Customized display of Explained statement history panel

To examine an access plan in detail, simply double-click on the Explained statement or highlight the entry of interest and use the panel menu to select **Statement ➔ Show access plan** on the Explained Statements History window.

All of the Explain statements will be displayed in the Explained Statements History list, but only the Explained statements with EXPLAIN SNAPSHOT information can be examined using Visual Explain.

Note

The Explain SQL option on the Control Center, or the Command Center are useful to Explain a single dynamic SQL statement.

You can add comments to the Explain snapshots listed in the Explained Statements History window. To add a comment describing a query, highlight the entry and then select **Statement ➔ Change**. This option can be used to provide a query tag, which can be used to help track the Explain snapshot information. You may also wish to remove Explain snapshots. The snapshots can be removed from the Explain tables by selecting **Statement ➔ Remove** after highlighting the entry to be removed.

The Visual Explain output displays a hierarchical graph representing the components of an SQL statement. Each part of the query is represented as a graphical object. These objects are known as *nodes*. There are two basic types of nodes:

- OPERATOR nodes indicate an action that is performed on a group of data.

- OPERAND nodes show the database objects where an operator action takes place. An operand is an object that the operators act upon. These database objects are usually tables and indexes.

There are many operators that can be used by the DB2 optimizer to determine the best access plan. Some of the operators used by Visual Explain are shown in Figure 33.

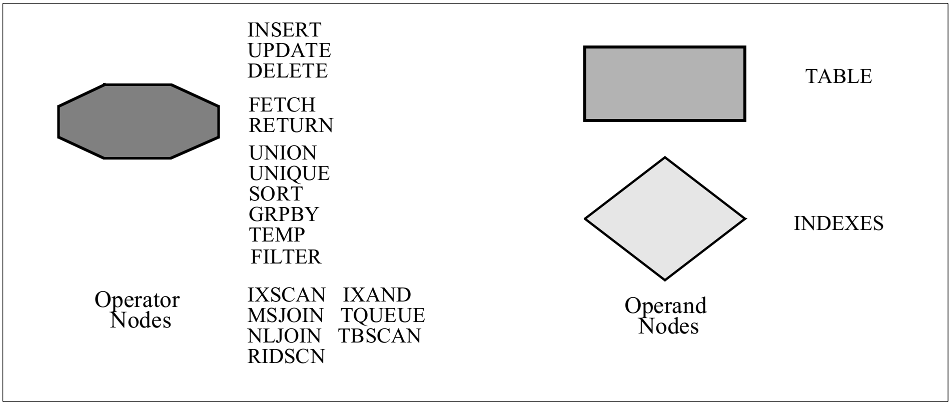
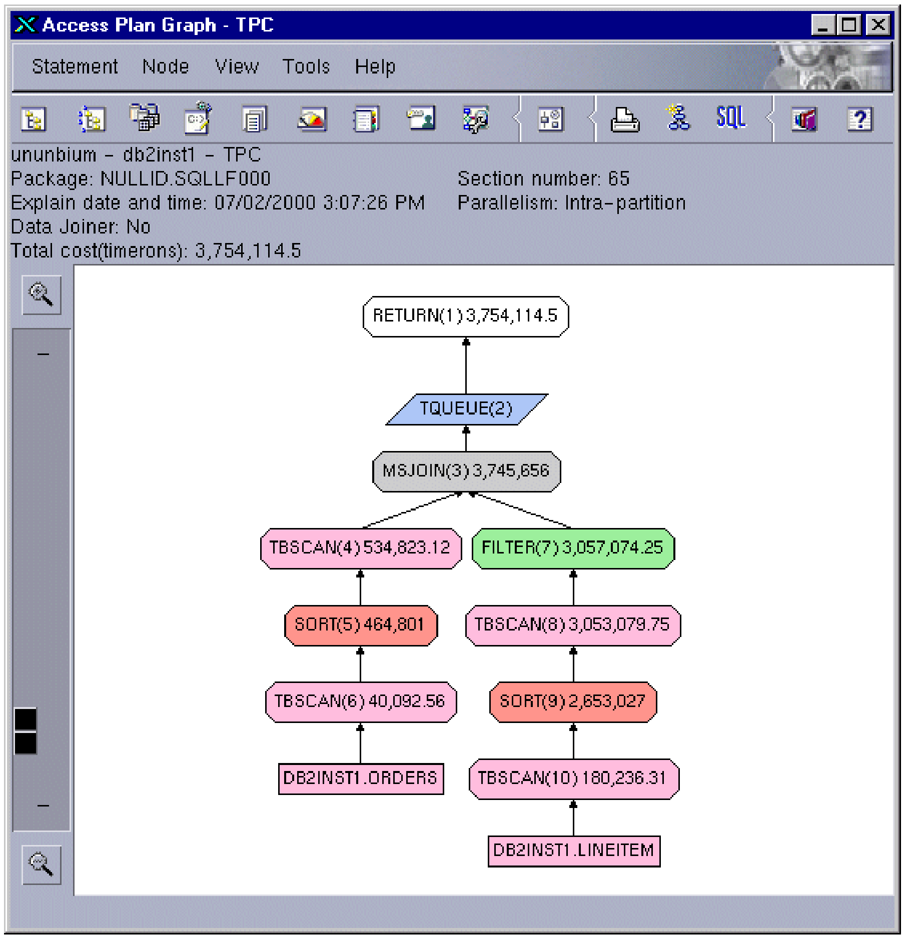


Figure 33. Operators and operands displayed in Visual Explain

These operators indicate how data is accessed (IXSCAN, TBSCAN, RIDSCN, IXAND), how tables are joined internally (MSJOIN, NLJOIN) and other factors, such as if a sort will be required (SORT). More information about the operators can be found using Visual Explain Online Help.

The objects shown in a Visual Explain graphic output are connected by arrows showing the flow of data from one node to another. The end of an access plan is always a RETURN operator.

The access plan shown in Figure 34 is a simple SQL statement: `SELECT * FROM LINEITEM,ORDRES WHERE L_EXTENDEDPRICE=O_TOTALPRICE`. In this example, there are two operands and ten operators. The operands are the `LINEITEM` table and the `ORDERS` table, and the operators include table scans (TBSCAN), sorts (SORT), a merge join (MSJOIN), a table queue (TQUEUE) and a RETURN operator.



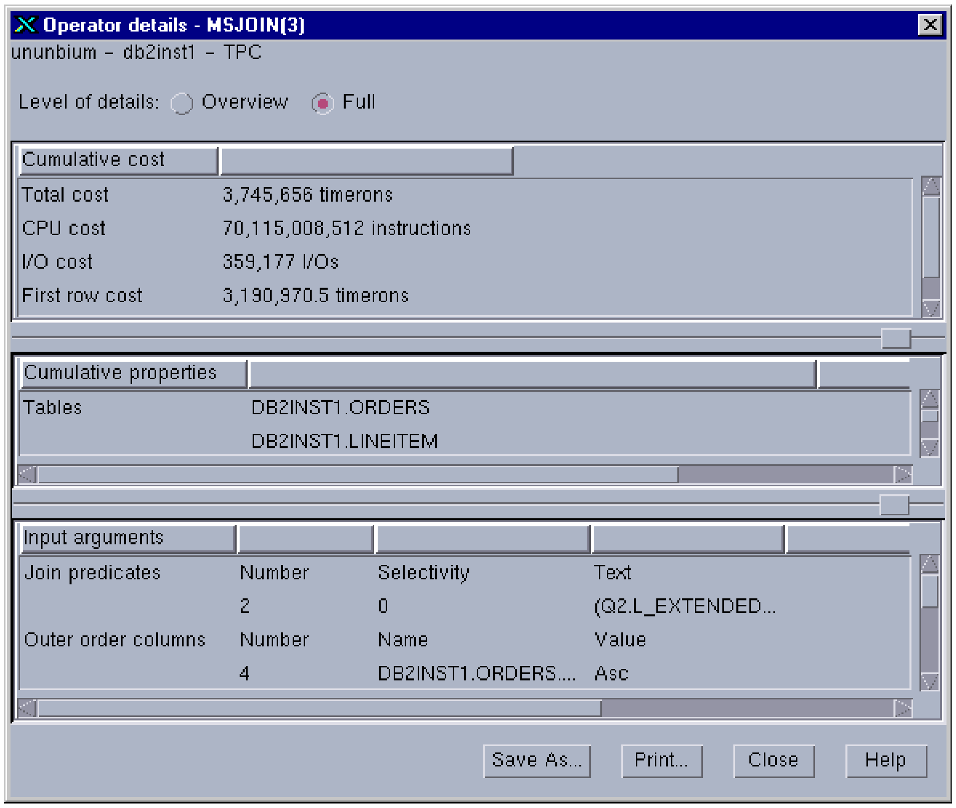
unumbium - db2inst1 - TPC
Package: NULLID.SQLLF000
Section number: 65
Explain date and time: 07/02/2000 3:07:26 PM
Parallelism: Intra-partition
Data Joiner: No
Total cost(timerons): 3,754,114.5

Figure 34. VIsual Explain: graphical access plan for SQL Statement

Generating Explain data for an SQL statement is the only way to analyze the access plan determined by the DB2 optimizer.

Each node, shown in an access plan graph, has detailed information that can be accessed by double-clicking on the node or by choosing the `Show details` option from the `Node` menu item.

To display the details of the merge join operation, select the `MSJOIN` operator node and then select `Show details` from the `Node` menu item. The information about the `MSJOIN` operation, shown in the access plan, is displayed in Figure 35.


Operator details - MSJOIN(3)

unumbium – db2inst1 – TPC

Level of details: Overview [✓ Full]

Cumulative cost	
Total cost	3,745,656 timerons
CPU cost	70,115,008,512 instructions
I/O cost	359,177 I/Os
First row cost	3,190,970.5 timerons

Cumulative properties	
Tables	DB2INST1.ORDERS
	DB2INST1.LINEITEM

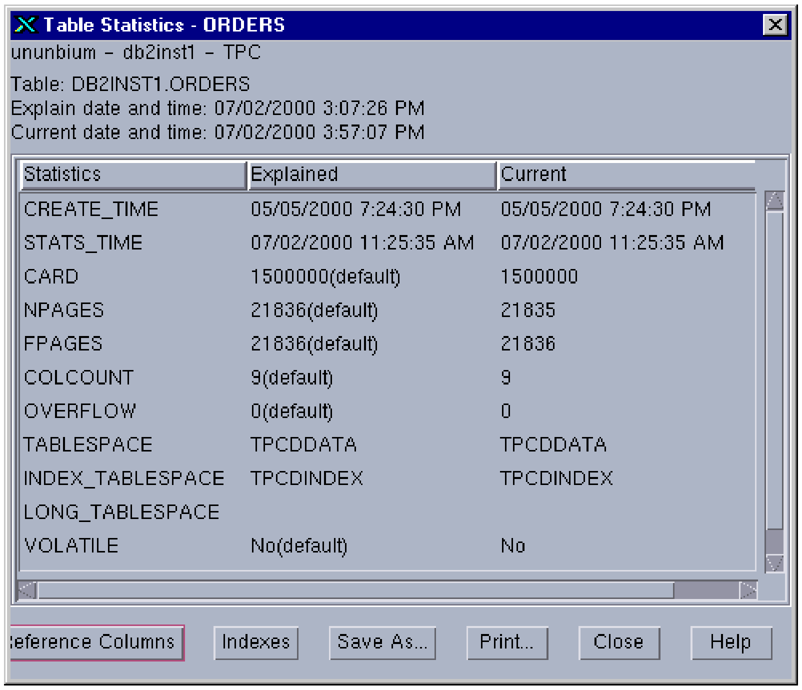
Input arguments			
Join predicates	Number	Selectivity	Text
	2	0	(Q2.L_EXTENDED...
Outer order columns	Number	Name	Value
	4	DB2INST1.ORDERS....	Asc

Figure 35. VIsual Explain: operator details

This window contains several different sections:

- **Cumulative costs** — Contains information about the estimated cumulative costs calculated using the statistics stored in the system catalog tables.
- **Cumulative properties** — Contains information about the table, columns, and so on, used to satisfy the query.
- **Input arguments** — Contains information about the input arguments that affect the behavior of the operator.

It is also possible to examine the detailed information about the operands. Select an operand node and then select **Show statistics** from the **Node** menu item. Figure 36 shows operand details for the ORDERS table.



unumbium - db2inst1 - TPC

Table: DB2INST1.ORDERS
Explain date and time: 07/02/2000 3:07:26 PM
Current date and time: 07/02/2000 3:57:07 PM

Statistics	Explained	Current
CREATE_TIME	05/05/2000 7:24:30 PM	05/05/2000 7:24:30 PM
STATS_TIME	07/02/2000 11:25:35 AM	07/02/2000 11:25:35 AM
CARD	1500000(default)	1500000
NPAGES	21836(default)	21835
FPAGES	21836(default)	21836
COLCOUNT	9(default)	9
OVERFLOW	0(default)	0
TABLESPACE	TPCDDATA	TPCDDATA
INDEX_TABLESPACE	TPCDINDEX	TPCDINDEX
LONG_TABLESPACE		
VOLATILE	No(default)	No

Figure 36. VIsual Explain: detailed statistics information for an operand

Detailed information for operand nodes shows the table or index statistics, including table space information, the number of columns, and the number of rows in the object. Figure 36 shows the Explained and current statistics from the system catalog tables. These statistics are used by the DB2 optimizer to determine the access plan.

When the optimizer has no statistics to work with for a table, or if the statistics for a table indicate that the cardinality of the table is relatively small, then the optimizer itself will attempt to calculate the cardinality of the table. The optimizer does this using certain factors including the average column length of the table and the number of pages used by the table.

Current statistics are the key to good access plans. If DB2 is not aware of the characteristics of objects involved in a query, it may not be able to generate a good access plan. To ensure that the latest statistics are available for the optimizer, a DB2 utility must be used. This utility is called RUNSTATS. Here is an example of gathering statistics for the DB2INST1.ORDERS table.

```
RUNSTATS ON TABLE DB2INST1.ORDERS
               WITH DISTRIBUTION AND DETAILED INDEXES ALL
```

Statistics for the DB2INST1.ORDERS table are stored in the system catalog tables. After running the RUNSTATS utility, rebind the packages against the database and re-Explain the SQL statement.

When determining the access plan for dynamic SQL statements, the DB2 optimizer always uses the current statistics. For a static SQL statement, DB2 uses the statistics available at BIND time (when the package was created). To ensure that current statistics are used with static SQL statements that were compiled before the statistics were updated, the packages must be recreated. This can be accomplished using the REBIND command.

Note

Updated statistics are always needed and become critical as your SQL statements grow in complexity

5.4.4.6 Guidelines on using EXPLAIN output

There are a number of ways in which analyzing the Explain data can help you to tune your queries and environment. For example:

Are indexes being used?

Creating appropriate indexes can have a significant benefit on performance. Using the Explain output, you can determine if the indexes you have created to help a specific set of queries are being used. In the Explain output, you should look for index usage in the following areas:

- Join Predicates
- Local Predicates
- GROUP BY clauses
- ORDER BY clauses
- The select list

You can also use the Explain Facility to evaluate whether a different index can be used instead of an existing index or no index at all. After creating a new index, collect statistics for that index using the RUNSTATS command and rebind packages for static SQL programs.

Over time, you may notice, through the Explain data, that instead of an index scan, a table scan is now being used for dynamic SQL statements. This can result from a change in the clustering of the table data. If the index that was previously being used now has a low cluster ratio, you want to:

- Reorganize your table to cluster the data according to that index.
- Use the `RUNSTATS` command to update the catalog statistics.
- Reexamine the Explain output to determine whether reorganizing the table has affected the access plan.

Is the type of access appropriate for the application?
You can analyze the Explain output and look for types of access to the data that, as a rule, are not optimal for the type of application being executed. For example:

- On-line Transaction Processing (OLTP) Queries
  OLTP applications are prime candidates to use index scans with range delimiting predicates because they tend to return only a few rows that are qualified using an equality predicate against a key column. If your OLTP queries are using a table scan, you may want to analyze the Explain data to determine the reasons why an index scan was not used.
- Browse-Only Queries
  The search criteria for a browse type query may be vague causing a large number of rows to qualify. If the user will usually only look at a few screens of the output data, you may want to try to ensure that the entire answer set need not be computed before some results are returned. In this case, the goals of the user are different from the basic operating principle of the optimizer, which attempts to minimize resource consumption for the entire query not just the first few screens of data.

5.4.5 The db2batch utility

The db2batch utility is a benchmarking tool that provides performance information. This tool processes batch SQL statements. When the db2batch tool is invoked, it will perform the following:

- Connect to the database
- Read, prepare, and execute the SQL statements
- Disconnect from the database
- Return the answer set, allowing you to determine the number of rows to be retrieved and the number of rows to be sent to output
- Return performance information, allowing you to specify the level of detail

• Return the mean values for “elapsed time” and “Agent CPU time” of all the SQL statements executed

The db2batch is usually fed by an input file. In this file, the user is able to set the different options and write the SQL statements that are to be executed by the utility. See the sample input file shown below:

```
--#SET perf_detail 2
--#SET rows_fetch 20
--#SET rows_out 10
select s_name from supplier;
--#SET rows_fetch -1
--#SET rows_out -1
select r_name,r_regionkey from region order by r_name;
```

When writing an input file, the basic rules are:

1. Options have the syntax --#SET <option> <value>. which lists the different options available.
2. Options apply only to the SQL statements below them.
3. Options can be “unset” by setting their value to -1.
4. SQL statements must be terminated by a semicolon.

Refer to Table 11 for the Snapshot Monitor Switch Groups.

Table 11. Snapshot monitor switch groups

Option	Value	Comment
perf_detail	0	No timing is to be done
	1	Return elapsed time only (default)
	2	Return elapsed time and CPU time
	3	Return a summary of monitoring information
	4	Return a snapshot for the database manager, the database, the application, and the statement (the latter is returned only if autocommit is off, and single statements, not blocks of statements, are being processed).
	5	Return a snapshot for the database manager, the database, the application, and the statement (the latter is returned only if autocommit is off, and single statements, not blocks of statements, are being processed). Also return a snapshot for the buffer pools, table spaces and FCM.

Option	Value	Comment
rows_fetch	-1 to n	Number of rows to be fetched from the answer set. The default value is -1 (all rows are to be fetched).
rows_out	-1 to n	Number of fetched rows to be sent to output. The default value is -1 (all fetched rows are to be sent to output).
sleep	1 to n	Number of seconds to sleep
delimiter		A one- or two-character end-of-statement delimiter. The default value is a semicolon (;)
timestamp		Generates a time stamp
pause		Prompts the user to continue

The output of the db2batch utility can be sent to a file. The level of detail, `perf_detail`, is set to 1 by default. This means that only the elapsed time for each SQL statement, agent CPU time for each SQL statement and the mean value of both will be returned. The default value for `rows_fetch` and `rows_out` is -1, meaning to fetch all rows from the answer set and to send all rows fetched to the output device.

Db2batch can be used to get snapshots easily. Setting `perf_detail` to 5 will get a complete snapshot (database manager, database and application) for every SQL statement included in the input file. Results include the same data elements used by the Snapshot Monitor. When benchmarking, setting `rows_out` to 0 will avoid “flooding” the output device with the rows fetched.

The following example shows the output from the db2batch tool with `perf_detail` 2:

```
select r_name,r_regionkey from region order by r_name

R_NAME                    R_REGIONKEY
-----------------------------------------
AFRICA                              0
AMERICA                             1
ASIA                                2
EUROPE                              3
MIDDLE EAST                         4


Number of rows retrieved is:       5
Number of rows sent to output is:  5

Elapsed Time is:           0.026      seconds


            *** FCM Snapshot ***

FCM buffers free                         = 1024
Low water mark of free buffers           = 996
FCM message anchors free                 = 384
Low water mark of free anchors           = 384
FCM connection entries free              = 384
        Low water mark of free connection entries    = 364
        FCM request blocks free                      = 480
        Low water mark of free request blocks        = 452

        Summary of Results
        ==================
                        Elapsed             Agent CPU         Rows      Rows
        Statement #     Time (s)            Time (s)          Fetched   Printed
        1                     0.026               0.010         5         5

        Arith. mean     0.026               0.01
        Geom.  mean     0.026               0.01
```

For more information on invocation syntax, and options, type db2batch -h on a command line.

5.4.5.1 db2bench.ksh

We have written a sample shell script to execute `db2batch` tool and store the output to the directory structure which all our sample scripts use (see 5.2, “Maintaining tuning information” on page 120). The source file of this script is included in Appendix A, “Sample scripts” on page 335.

5.4.6 CLI/ODBC/JDBC Trace Facility

The CLI/ODBC/JDBC Trace Facility of DB2 is an essential tool for problem determination and general understanding of your applications using CLI and others which use the DB2 CLI driver (for example, ODBC, JDBC, and SQLJ applications). All function calls executed are recorded in a text file for later analysis. In addition to functional information, however, the trace file contains elapsed time information which can be extremely useful for application and database tuning.

5.4.6.1 Getting started

Let us review how you can obtain a CLI/ODBC/JDBC trace. Full information can be found in Appendix K of the *DB2 UDB Call Level Interface Guide and Reference,* SC09-2950.

First of all, be sure to distinguish CLI/ODBC/JDBC tracing from ODBC tracing. An ODBC trace shows the calls from the application to an ODBC Driver Manager. A CLI/ODBC/JDBC trace shows the calls made to CLI, either directly from the application or from an ODBC Driver Manager. It is sometimes useful to obtain both traces, but only CLI tracing is discussed in this section.

To obtain a CLI trace, run the application after using one of these means to activate tracing:

- Command Line Processor: See the following example:

  ```
  db2 update cli cfg for section common using trace 1 tracepathname <fully
  qualified pathname>
  db2 update cli cfg for section common using tracecomm 1
  ```

- Client Configuration Assistant (OS/2 and Windows only): Select a database (it must be one registered for ODBC), then Properties, CLI/ODBC Settings, Advanced, and Service.

The `TRACEPATHNAME` keyword specifies the directory path name used to store individual trace files.

The `TRACECOMM` keyword specifies whether the network request information is included in the trace file. For the `TRACECOMM` keyword, you can specify 0 or 1. For the default setting of 0, no network request information is captured. Setting 1 for this keyword significantly changes the trace output, as described later.

You can also set the following parameters:

- `TRACEFLUSH` keyword
  This keyword specifies whether a write to disk is forced after each CLI/ODBC entry. For the default setting of 0, a write is not performed after every entry. Setting `TRACEFLUSH=1` has a large performance impact; use only if the application may not exit normally.
- `TRACETIMESTAMP` keyword
  This keyword specifies whether a timestamp is added at the beginning of each line, as described later.
- `TRACEPIDTID` keyword
  This keyword causes each line to begin with the process ID and thread ID of the application thread issuing the corresponding call.

5.4.6.2 CLI trace file contents

The *Call Level Interface Guide and Reference*, SC09-2950 contains a full description of the contents of a CLI trace file. The `TRACECOMM` keyword was introduced in DB2 UDB Version 5.2 and causes information to be generated in addition to what is produced by a “regular” trace. The `TRACETIMESTAMP` and `TRACEPIDTID` keywords are new in Version 7.1 and provide additional timing and thread identification information. We will now discuss what each type of trace contains, paying particular attention to `TRACECOMM` output and the new Version 7.1 keywords.

Regular trace contents

The CLI driver writes trace records when it is entered and when it exits, to reflect the activity just completed. Thus, in the following code snippet, the first `SQLDisconnect` record (2) is written on entry to the CLI driver, as are all records marked by “--->”. The “`Time elapsed`” (+8.430000E-004 seconds = 0.0008 seconds) represents the elapsed time in the application between the last exit from the CLI driver (after the `SQLFreeHandle` call) and the re-entry to process the `SQLDisconnect` call. So, if you see long elapsed times in the “--->” records, it could reflect a performance problem or heavy activity in the application, or, conversely, idle time while waiting for user input to be supplied.

```
(1) SQLFreeHandle( )
        <--- SQL_SUCCESS   Time elapsed - +2.758000E-003 seconds

(2) SQLDisconnect( hDbc=0:1 )
        ---> Time elapsed - +8.430000E-004 seconds

(3) SQLDisconnect( )
        <--- SQL_SUCCESS   Time elapsed - +1.001400E-002 seconds
```

The second `SQLDisconnect` record (3) was written on exit from the CLI driver, and its "`Time elapsed`" (+1.001400E-002 seconds = 0.01 seconds) represents the elapsed time in DB2 to process the `SQLDisconnect` call. As do all records marked by "<---" (which is followed by the return code of the call), this time includes time in the CLI driver, the DB2 runtime client (formerly known as the CAE), the entire communication infrastructure between the client and the database server, and the time in the database server itself. (We will use the phrase "in DB2" to refer to these components collectively.)

As useful as these elapsed times are (and later we'll discuss how to maximize their usefulness), it is often more interesting to obtain the additional information that comes with `TRACECOMM`.

TRACECOMM trace contents

There are three main reasons to use `TRACECOMM=1`:

- To find out when a client-to-server communication occurs, either locally or over the network. Many CLI functions are processed completely on the client, and it makes sense to pay less attention to them than to the calls involving communication: (a) network requests typically have a cost of at least an order of magnitude higher than requests that are processed on the client only; (b) generally, there is less that can be done to affect the execution time of client-only calls.
- To find out the number of bytes sent and received in each communication.
- To break down CLI call elapsed times into their components: (a) in-CLI and (b) in-communication-and-server.

Let us now look at how the trace records in the above example will change if the fetch program is run again, this time with `TRACECOMM` activated (=1). (Record numbers on the left in the trace records below were added to aid in the discussion.)

```
 SQLFreeHandle( )
(1)    <--- SQL_SUCCESS   Time elapsed - +2.894000E-003 seconds

   SQLDisconnect( hDbc=0:1 )
(2)    ---> Time elapsed - +1.587000E-003 seconds
(3)    sqlccsend( ulBytes - 72 )
(4)    sqlccsend( Handle - 539269544 )
(5)    sqlccsend( ) - rc - 0, time elapsed - +1.960000E-004
       sqlccrecv( )
(6)    sqlccrecv( ulBytes - 27 ) - rc - 0, time elapsed - +4.278810E-001
              SQLDisconnect( )
(7)     <--- SQL_SUCCESS   Time elapsed - +4.296480E-001 seconds
```

The obvious change is in the additional lines under the `SQLDisconnect` entry record. The existence of these lines confirms that a server communication occurred to process the `SQLDisconnect`.

Other points to note about these new lines are as follows:

- Line (2), "`---> Time elapsed ...`" indicates that approximately 1 millisecond was spent in the application. So, the application time in this line has the same meaning as in a trace file generated without `TRACECOMM`.
- Line (3), "`sqlccsend( ulBytes - 72 )`" indicates that 72 bytes were sent to the server.
- Line (4), "`sqlccsend( Handle - 539269544 )`" indicates that the send was to be a thread with the handle ID indicated.
- Line (5), "`sqlccsend( ) - rc - 0 ...`" indicates that the send was successful and had an elapsed time of approximately 0.2 milliseconds. Note that these sends are done asynchronously, so they should always have a very small elapsed time. CLI then waits for the response to come back.
- Line (6), "`sqlccrecv( ulBytes - 27 ) ...`" indicates that 27 bytes were received from the server. It also indicates that 0.4296 seconds were spent from the completion of the send to the completion of the receive.

Note that some functions, such as `SQLConnect`, can execute multiple send-receive pairs, and there will be an entry for each send and receive.

To see the derivation of the elapsed times in a different way, refer to the following diagram (Figure 37). It shows the main components involved in executing the calls in the previous trace snippet. The arrows represent the flow of control as time passes (going down the page). The letters (a), (b), and so on, represent the points at which the DB2 CLI driver records timestamp information:

- The elapsed time in line (2) above is the difference between times (a) and (b). This is the time spent in the application between the return to the application after the `SQLFreeHandle` call was processed, and the entry to the CLI Driver to process the `SQLDisconnect` call. (Actually, it also includes the communication time from and to the CLI driver, and the time in the driver between when the timestamp was taken and the exit or entry, but those times are generally negligible.)
- The time in line (5) is the difference between (c) and (d), or the Send time.
- The time in line (6) is the difference between (d) and (e), or the Server and Receive time.

- The time in line (7) is the difference between (b) and (f), or the “DB2 time”.

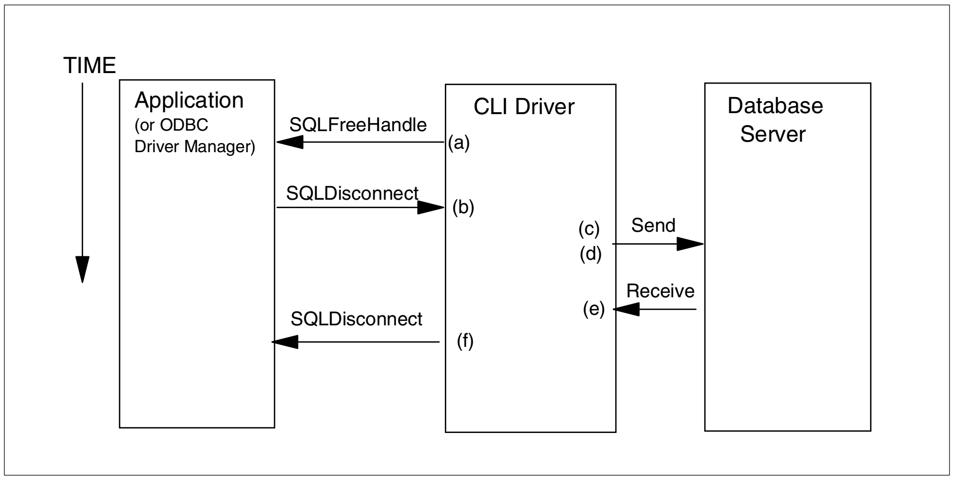


Figure 37. CLI calls

Timestamps and process/thread IDs in trace

In Version 7.1 (and Version 6.1 with FixPak 4) two new trace keywords were added:

- TRACETIMESTAMP. This causes a timestamp to be added at the beginning of each line. There are three formats available, with the default value being 0 (off):
- =1 [<number of seconds since 01/01/1970>.<microseconds>-<formatted timestamp>]
- =2 [<number of seconds since 01/01/1970>]
- =3 [<formatted timestamp>]
- TRACEPIDTID. This causes each line to begin with the process ID and thread ID of the application thread issuing the corresponding call.

Following are some sample outputs for the various combinations of values for `TRACETIMESTAMP` and `TRACEPIDTID`. Note that the regular trace output is unchanged, but is shifted to the right of the new information.

```
*********************
* TraceTimestamp=1  *
* TracePidTid=1     *
*********************
[ Process: 298, Thread: 317 ]
[ Date & Time:          02-11-2000 09:49:07.000048 ]
(lines omitted here in this and the other examples)

[0000000298 0000000317] [950280547.000306 - 02-11-2000 09:49:07.000306]
SQLSetEnvAttr( )
[0000000298 0000000317] [950280547.000393 - 02-11-2000 09:49:07.000393]
<--- SQL_SUCCESS   Time

[0000000298 0000000317] [950280547.162111 - 02-11-2000 09:49:07.162111]
SQLAllocConnect( hEnv=0:1,
[0000000298 0000000317] [950280547.162256 - 02-11-2000 09:49:07.162256]
---> Time elapsed - +1.6343

[0000000298 0000000317] [950280547.336041 - 02-11-2000 09:49:07.336041]
SQLAllocConnect( phDbc=0:1 )
[0000000298 0000000317] [950280547.336164 - 02-11-2000 09:49:07.336164]
<--- SQL_SUCCESS   Time ela

*********************
* TraceTimestamp=1  *
* TracePidTid=0     *
*********************
[ Process: 298, Thread: 317 ]
[ Date & Time:          02-11-2000 09:49:14.000025 ]

[950280554.000182 - 02-11-2000 09:49:14.000182] SQLSetEnvAttr( )
[950280554.000246 - 02-11-2000 09:49:14.000246]     <--- SQL_SUCCESS
Time elapsed - +6.400000E-005

*********************
* TraceTimestamp=2  *
* TracePidTid=0     *
*********************
[950280554.000183] SQLSetEnvAttr( )
[950280554.000213]     <--- SQL_SUCCESS   Time elapsed - +3.000000E-005 seconds

*********************
* TraceTimestamp=3  *
* TracePidTid=0     *
*********************
[02-11-2000 09:49:15.000184] SQLSetEnvAttr( )
[02-11-2000 09:49:15.000233]     <--- SQL_SUCCESS   Time elapsed - +4.900000E-005 secon
```

5.4.6.3 Analysis objectives

You may have specific objectives in mind when analyzing a CLI trace, or you may just be looking to see if anything interesting appears. Here are some of the most common objectives (from a performance point of view), which are discussed individually in the next section:

- Get a breakdown of time spent in the application versus time spent in DB2.
- Find out how long a particular CLI call is taking.
- Focus on those CLI calls which are typically where performance problems are experienced.
- Find the longest intervals of execution, in either the application or in DB2.
- Find the number of CLI calls of various types.
- Find out how much data is transferred to or from the server.
- Study the timing relationships between multiple threads in an application.

Some other typical objectives involve determining which CLI calls are occurring (a third-party tool could be generating them), or to look for execution errors, but these objectives are beyond the scope of this section.

Analyzing the trace and finding the problem

Depending on your objectives, the analysis of the trace file can be attacked in various ways and with various tools to extract interesting information.

Application time versus DB2 time

Before embarking on a journey to improve a DB2 application's performance, it is important to know where the majority of time is being spent. For some applications, the vast majority of the elapsed time is spent in the application, not in DB2. In such cases, spending time on DB2 tuning is essentially a wasted exercise, at least until the application problem is dealt with.

Application and DB2 times can be summarized very easily using a Java tool called `CLITraceParser`, which is available at:
`ftp://ftp.software.ibm.com/ps/products/db2/tools/` in the file `CLITraceParser.zip`. See the `README.TXT` file in that tool subdirectory for information on installation and usage. Note that `CLITraceParser` replaces and improves upon `parseCLITrace`, which is also available at that FTP site.

The CLITraceParser parses a CLI trace and produces a summary of it. Here is the output of the tool for a sample CLI trace file:

```
 CLI Trace Report generated by CLITraceParser
==========================================================

CLI Trace file                  : casestudy1_TraceComm_Off
Lines read in file              : 108
Trace build info                : null

Overall Trace statistics
========================================================

             14 statements in trace.
         93.110 seconds total trace time.
          0.012 seconds spent for application processing.
         93.098 seconds spent for CLI processing.

Network Specific CLI processing time statistics
========================================================
              0 network flows sent to transmit
              0 bytes, requiring a total of
          0.000 seconds.

              0 network flows received, transmitting
              0 bytes, requiring a total of
          0.000 seconds.

End of overall trace statistics report

**************************************************************************

Function specific statistics
========================================================

                      Timing             Network Send      Network Receive
Function Name   Total Application CLI    Flows Bytes Time  Flows Bytes Time
------------------------------------------------------------------------
SQLSetConnectAttr 1   0.001       0.001  0     0    0.000  0     0    0.000
SQLExecDirect     1   0.000       92.204 0     0    0.000  0     0    0.000
SQLBindCol        1   0.000       0.000  0     0    0.000  0     0    0.000
SQLFetch          2   0.001       0.002  0     0    0.000  0     0    0.000
SQLFreeHandle     3   0.001       0.005  0     0    0.000  0     0    0.000
SQLAllocHandle    3   0.008       0.017  0     0    0.000  0     0    0.000
SQLConnect        1   0.000       0.856  0     0    0.000  0     0    0.000
SQLDisconnect     1   0.001       0.010  0     0    0.000  0     0    0.000
SQLEndTran        1   0.000       0.002  0     0    0.000  0     0    0.000

End of function specific statistics report

**************************************************************************

Report of errors that occurred in this CLI Trace
================================================
No errors in trace.

End of error report.

==========================================================
                End of CLI Trace Report
```

The output is quite self-explanatory:

- The first section ("Overall Trace statistics") shows the total trace time and the breakdown of time between application and DB2.
- The next section ("Network Specific CLI processing time statistics") summarizes the number of sends and receives and the total bytes transmitted.
- The final section ("Function specific statistics") shows the number of calls by function, each function's elapsed time in application and DB2, and the related network activity (if `TRACECOMM=1` was specified).

The output clearly indicates that time in DB2 is the key component of elapsed time to be addressed in this particular example.

The following example shows the output of `CLITraceParser` for a trace file taken from the same application as for the summary above, but this time with `TRACECOMM=1`. (A different run was made to collect this trace, so there are small differences in elapsed times.)

```
CLI Trace Report generated by CLITraceParser
========================================================

CLI Trace file                    : casestudy1_TraceComm_On
Lines read in file                : 133
Trace build info                  : null

Overall Trace statistics
======================================================

            14 statements in trace.
        88.332 seconds total trace time.
         0.014 seconds spent for application processing.
        88.319 seconds spent for CLI processing.

Network Specific CLI processing time statistics
======================================================
             5 network flows sent to transmit
          2762 bytes, requiring a total of
         0.001 seconds.

             5 network flows received, transmitting
          1935 bytes, requiring a total of
        88.068 seconds.

End of overall trace statistics report

*************************************************************************

Function specific statistics
======================================================

                      Timing             Network Send      Network Receive
Function Name   Total Application CLI    Flows Bytes Time  Flows Bytes Time
-------------------------------------------------------------------------
SQLSetConnectAttr 1   0.001       0.001  0     0     0.000 0     0     0.000
SQLExecDirect     1   0.000       87.136 1     324   0.000 1     382   87.130
SQLBindCol        1   0.000       0.000  0     0     0.000 0     0     0.000
SQLFetch          2   0.001       0.001  0     0     0.000 0     0     0.000
SQLFreeHandle     3   0.001       0.005  0     0     0.000 0     0     0.000
SQLAllocHandle    3   0.009       0.017  0     0     0.000 0     0     0.000
SQLConnect        1   0.000       1.150  2     2170  0.000 2     1499  0.933
SQLDisconnect     1   0.001       0.005  1     72    0.000 1     27    0.003
SQLEndTran        1   0.000       0.003  1     196   0.000 1     27    0.002

End of function specific statistics report

*************************************************************************

Report of errors that occurred in this CLI Trace
================================================
No errors in trace.

End of error report.

========================================================
                End of CLI Trace Report
```

Note that the information in the report is the same as with `TRACECOMM=0`, except that the network statistics, which were previously zero, are now supplied.

Finding how long a particular CLI call took

This task is trivial, requiring only that you can find the trace record of interest. Often this can be done by searching for the CLI call type. At other times you may be interested in a particular SQL statement, and you can search for it using a search string such as `SELECT`, or a table name, or other text that you know occurs within the statement.

Focusing on typical problem calls

In general, most of the performance problems in CLI applications occur in a relatively small subset of calls. Focusing on these functions will allow you to get a sense of what the database requests are and how the data is flowing between the application and the server. A close look at these and other CLI functions comes later. We suggest that you look particularly at this set of calls:

- SQLConnect or SQLDriverConnect — Connect to a database.
- SQLExecDirect — Combined prepare and execute of an SQL statement.
- SQLPrepare and SQLExecute — Prepare and Execute done separately to allow many execute requests for the same statement, using only one prepare.
- SQLFetch — Fetch data either from the local data block or from the server. In the case of a blocking cursor, most fetches are handled locally. Data is either retrieved into bound application variables, or into CLI memory, ready for SQLGetData calls.
- SQLGetData — Copy (and convert if required) data from CLI to the application.
- SQLError — Some applications may continually generate warnings or errors, but you can treat them as “normal”. Note that the `CLITraceParser` lists all of the errors in its output file for a given trace.

Finding the longest-running events

Finding the longest-running events, particularly in DB2, is a very common and important task, and the use of appropriate tools can speed up the process significantly. If the trace is small, you can simply page through the file, but usually a better alternative is to edit the file and search for the desired strings. For files that are too large to be edited, you can use the `grep` command to extract occurrences of the desired strings.

Some strings of particular interest to search for are:

- **E+** : Occurs in all events of 1 second or longer (or zero)
- **E+000** : Occurs in all events of 1 second to 9.9 seconds (or zero)
- **E+001** : Occurs in all events of 10 seconds to 99.9 seconds
- **E-** : Occurs in all events of less than 1 second (except zero)
- <--- : Flags times in DB2 (does not appear in TRACECOMM-specific entries)
- **--->** : Flags times in the application

Search examples
Following are some examples of commands to do specific searches.
In each one, “trace_file” must be replaced by the name of the trace file being analyzed.

Note that trace file lines that are specific to `TRACECOMM` contain “`E+`” or “`E-`”, but not “<---” or “--->”. Therefore, the first example will find TRACECOMM specific entries; but all of the other examples, because they include a search for an “arrow”, will not:

- The following can be used on a Windows command line to list all elapsed times, both in DB2 and the application, along with their line numbers in the trace file:

  ```
  findstr /n "E+ E-" trace_file | more
  ```

- The following can be used on a Windows command line to list elapsed times of 10 to 999.9 seconds in DB2, with their line numbers in the trace file:

  ```
  findstr /n "E+001 E+002" trace_file | findstr /c:"<---"
  ```

- The following can be used on a Windows command line to list elapsed times of 0.10 to 0.99 seconds in the application, with their line numbers in the trace file:

  ```
  findstr /n "E-001" trace_file | findstr /c:"--->"
  ```

- The sort command in Windows is quite limited, compared to sort in UNIX. The following can be used on UNIX, or on Windows with the MKS toolkit (or possibly via other tools from freeware or shareware sources), to list, in descending order and with line numbers within the trace file, the 20 longest elapsed times in DB2 that are one second or longer.

  ```
  grep -n -e "<---" trace_file | grep -e "E+" | sort
    -k 6.12b,6rn -k 6.2b,6.9brn | head -n 20
  ```

- This slight variation of the previous command gives the 20 longest DB2 times under one second:

```
grep -n -e "<---" trace_file | grep -e "E-" | sort
   -k 6.12b,6n -k 6.2b,6.9brn | head -n 20
```

Note that "`E+`" is changed to "`E-`", and "`r`" is omitted from "`-k 7.12b,7rn`". The 6's in the commands indicate that the 6th token in each line is to be used as a sort key. If the "-n" option had not been specified in the `grep` command, there would not be a line number in each grep output line (one less token per line), and so the 6's in the sort commands would have to be changed to 5's. Similar adjustments need to be made if non-zero values of the `TRACETIMESTAMP` or `TRACEPIDTID` keywords are specified when generating the trace.

Finding the numbers of calls

In some scenarios, it is not the elapsed times of individual CLI calls that is the problem, but the fact that hundreds or even thousands of calls may be occurring for a given application task. Obviously, even very short calls can become a performance burden if executed enough times.

The `CLITraceParser` tool, discussed in "Application time versus DB2 time" on page 183, is tailor-made for addressing this issue. It provides the number of calls to each function and the total time for each.

Finding the amount of data transferred to or from the server

The amount of data transferred between the application and the server can be a key factor in application response times. Determining the amount of data transferred is easily accomplished by using the `TRACECOMM` option, as discussed in , "TRACECOMM trace contents" on page 179.

Analyzing timing relationships between multiple threads

It can be difficult to relate the behavior of different threads in an application, but using CLI trace options can make this much easier. First of all, recall that `TRACEPATHNAME` is used to name the path in which a separate trace file for each thread will be created. Using `TRACETIMESTAMP` and `TRACEPIDTID`, you can have timestamp and process/thread information added to each trace entry.

The following example shows how you can use the sort command on UNIX or on Windows with the MKS toolkit (or possibly via other tools from freeware or shareware sources) to merge the trace entries for multiple files, sort them by timestamp, and write them to a new file, allowing you to follow the sequence of events in all of the threads:

```
Trace for PID = 298, TID = 317, in file 000298.317

[0000000298 0000000317] [950280547.000306 - 02-11-2000 09:49:07.000306]
SQLSetEnvAttr( )
[0000000298 0000000317] [950280547.000393 - 02-11-2000 09:49:07.000393]
<--- SQL_SUCCESS   Time

[0000000298 0000000317] [950280547.162111 - 02-11-2000 09:49:07.162111]
SQLAllocConnect( hEnv=0:1,
[0000000298 0000000317] [950280547.162256 - 02-11-2000 09:49:07.162256]
---> Time elapsed - +1.6343

[0000000298 0000000317] [950280547.536041 - 02-11-2000 09:49:07.536041]
SQLAllocConnect( phDbc=0:1 )
[0000000298 0000000317] [950280547.536164 - 02-11-2000 09:49:07.536164]
<--- SQL_SUCCESS   Time ela

Trace for PID = 298, TID = 318, in file 000298.318

[0000000298 0000000318] [950280547.000612 - 02-11-2000 09:49:07.000612]
SQLSetEnvAttr( )
[0000000298 0000000318] [950280547.001393 - 02-11-2000 09:49:07.001393]
<--- SQL_SUCCESS   Time

[0000000298 0000000318] [950280547.262111 - 02-11-2000 09:49:07.262111]
SQLAllocConnect( hEnv=0:1,
[0000000298 0000000318] [950280547.262256 - 02-11-2000 09:49:07.262256]
---> Time elapsed - +1.6343

[0000000298 0000000318] [950280547.336041 - 02-11-2000 09:49:07.336041]
SQLAllocConnect( phDbc=0:1 )
[0000000298 0000000318] [950280547.336164 - 02-11-2000 09:49:07.336164]
<--- SQL_SUCCESS   Time ela

Merged file 000298.mrg, created by "sort -k 3.2n 000298.* > 000298.mrg" on Unix

[0000000298 0000000317] [950280547.000306 - 02-11-2000 09:49:07.000306]
SQLSetEnvAttr( )
[0000000298 0000000317] [950280547.000393 - 02-11-2000 09:49:07.000393]
<--- SQL_SUCCESS   Time
[0000000298 0000000318] [950280547.000612 - 02-11-2000 09:49:07.000612]
SQLSetEnvAttr( )
[0000000298 0000000318] [950280547.001393 - 02-11-2000 09:49:07.001393]
<--- SQL_SUCCESS   Time
[0000000298 0000000317] [950280547.162111 - 02-11-2000 09:49:07.162111]
SQLAllocConnect( hEnv=0:1,
[0000000298 0000000317] [950280547.162256 - 02-11-2000 09:49:07.162256]
---> Time elapsed - +1.6343
[0000000298 0000000318] [950280547.262111 - 02-11-2000 09:49:07.262111]
SQLAllocConnect( hEnv=0:1,
[0000000298 0000000318] [950280547.262256 - 02-11-2000 09:49:07.262256]
---> Time elapsed - +1.6343
[0000000298 0000000318] [950280547.336041 - 02-11-2000 09:49:07.336041]
SQLAllocConnect( phDbc=0:1 )
[0000000298 0000000318] [950280547.336164 - 02-11-2000 09:49:07.336164]
<--- SQL_SUCCESS   Time ela
[0000000298 0000000317] [950280547.536041 - 02-11-2000 09:49:07.536041]
SQLAllocConnect( phDbc=0:1 )
[0000000298 0000000317] [950280547.536164 - 02-11-2000 09:49:07.536164]
<--- SQL_SUCCESS   Time ela
```

You can also use this approach to merge trace files for different applications, or for different invocations of the same application, but be careful: if multiple clients are involved, each one will have a slightly (or greatly) different system time that will be used for the timestamps, so the merged sequence of events may not reflect the true sequence.

One thing you may see in the merged trace file is one thread not having any entries for a long period. This could simply indicate that a long-running task was being executed, but it could also indicate that the thread is in a lock wait state. Note that an application's multiple threads can have multiple connections to a database, and from the database server's point of view there is no special relationship between those connections. For instance, they can lock each other out, even though they belong to the same application and are executed through the same authorization ID.

Chapter 6. Tuning configuration parameters

In this chapter, we discuss how memory is used by DB2 UDB, and then introduce configurable parameters of DB2 UDB to optimize the performance of your database. You can see more than one hundred configurable parameters in the manual *DB2 UDB Administration Guide - Performance*, SC09-2945, although you do not have to tune all of them. You can focus on a few important parameters which highly impact the database performance only. Here we will introduce some parameters, grouped according to which areas of system resource usage these parameters affect.

As you have seen in Chapter 2, “Setting up the Control Center” on page 15, when you create a new database, you should use the Configure Performance Wizard to obtain the recommended values of the performance related configuration parameters. You should also use this wizard to obtain new recommended values when your database has been significantly updated. You can start with the recommendations made by this wizard, and then make further adjustments to optimize the performance of your database. This chapter is intended to aid you in making those further adjustments.

6.1 Configuration parameters

DB2 was designed with tuning and configuration parameters that fall into two general categories:

- Database manager configuration parameters
- Database configuration parameters

6.1.1 Database manager configuration parameters

Each instance of the database manager has a set of the database manager configuration parameters (also called database manager parameters). These affect the amount of system resources that will be allocated to a single instance of the database manager. Also, they configure the setup of the database manager and the different communications subsystems (such as TCP/IP and APPC) based on environmental considerations. In addition, there are other database manager configuration parameters that serve informative purposes only and cannot be changed. All of these parameters effect the instance level and have global applicability independent of any single database stored under that instance of the database manager.

To view, set, and reset the database manager configuration parameters, you can use the following methods:

- Control Center

  The Control Center provides the Configure Instance notebook which you can use to view, set, and reset the database manager configuration parameters. See 2.3.1, "Setting up configuration parameters" on page 22.

- Database parameters

  From the Command Line Processor or the Command Center, you can execute these commands:

  ```
  GET DBM CFG
  UPDATE DBM CFG
  RESET DBM CFG
  ```

6.1.2 Database configuration parameters

Each database has a set of the database configuration parameters (also called database parameters). These affect the amount of system resources that will be allocated to that database. In addition, there are some database configuration parameters that provide descriptive information only and cannot be changed; others are flags that indicate the status of the database.

To view, set, and reset the database configuration parameters, you can use the following methods:

- Control Center

  The Control Center provides the Configure Database notebook which you can use to view, set, and reset the database configuration parameters. See 2.3.1, "Setting up configuration parameters" on page 22.

- Database Parameters

  From the Command Line Processor or the Command Center, you can execute these commands:

  ```
  GET DB CFG FOR database_name
  UPDATE DB CFG FOR database_name USING parameter_name value
  RESET DB CFG FOR database_name
  ```

6.2 Memory model

In Chapter 1 we showed an overview of the architecture and processes of DB2 UDB. Before discussing each configuration parameter, we introduce you to the memory model of DB2, because many of the configuration parameters available in DB2 affect memory usage on the system. You should understand how memory is divided among the different heaps before tuning to balance overall memory usages on the system.

6.2.1 Types of memory used by DB2 UDB

Figure 38 shows that the database manager uses different types of memory. In this figure, we assume that intra-partition parallelism is enabled.

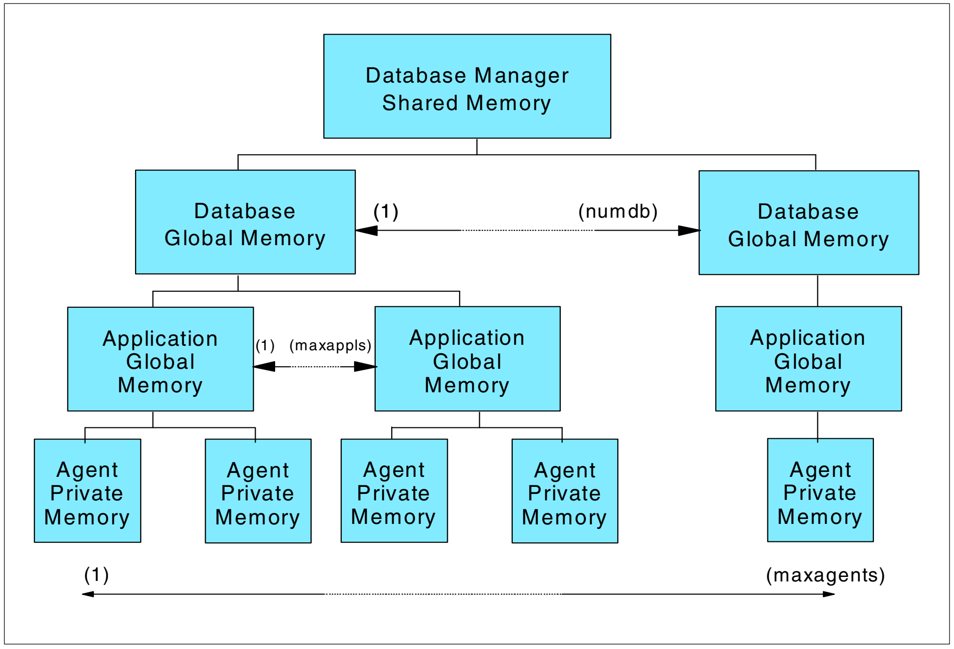


Figure 38. Memory segments used by DB2 UDB

The Database Manager Shared Memory is allocated when the database manager is started using the `db2start` command, and remains allocated until the database manager is stopped using the `db2stop`. This memory is used to manage activity across all database connections. From the Database Manager Shared Memory, all other memory is attached/allocated.

The Database Global Memory (also called Database Shared Memory) is allocated for each database when the database is activated using the `ACTIVATE DATABASE` command or the first application connects to the database. The Database Global Memory remains allocated until the database is deactivated using the `DEACTIVATE DATABASE` command or the last application disconnects from the database. The Database Global Memory contains memory areas such as buffer pools, lock list, database heap and utility heap. The database manager configuration parameter `NUMDB` defines the maximum number of concurrent active database. If the value of this parameter increases, the number of Database Global Memory segments may grows depending on the number of active databases.

The Application Global Memory is allocated for each connection when the connection is established to a database and remains allocated until the connection is terminated. This memory is used by DB2 agents, including coordinator agents and subagents working on behalf of the application, to share data and coordinate activities among themselves.

The Agent Private Memory is allocated for each DB2 agent when the DB2 agent assigned to work for an application. The Agent Private Memory contains memory areas which will be used only by this specific agent, such as sort heaps and application heaps.

The Agent Private Memory remains allocated even after the DB2 agent completes tasks for the application and gets into idle state. However, if you set the DB2 registry variable `DB2MEMDISCLAIM` to `YES`, then DB2 disclaims some or all memory once freed, depending on the value given with the DB2 registry variable `DB2MEMMAXFREE` which defines the amount of the memory to be retained by each DB2 agent. We discuss these registry variables later in this chapter.

The database configuration parameter `MAXAPPLS` defines the maximum number of applications that can simultaneously connect to the database. The database manager configuration parameter `MAXAGENTS` defines the maximum number of DB2 agents including coordinator agents and subagents in the instance. If the value of this parameter increases, the number of the Application Global Memory segments and the Agent Private Memory segments may grow, depending on the number of connected applications and DB2 agents, respectively.

Note

Application Global Memory is allocated if you enable intra-partition parallelism, or if the database manager is in a partitioned database environment using DB2 UDB Enterprise-Extended Edition, which is beyond our discussion

6.2.2 How memory is used

Figure 39 and Figure 40 show how memory is used to support applications. In the previous section we introduced some configuration parameters which may affect the number of memory segments. We introduce the configuration parameters which allow you to control the size of each memory by limiting their size.

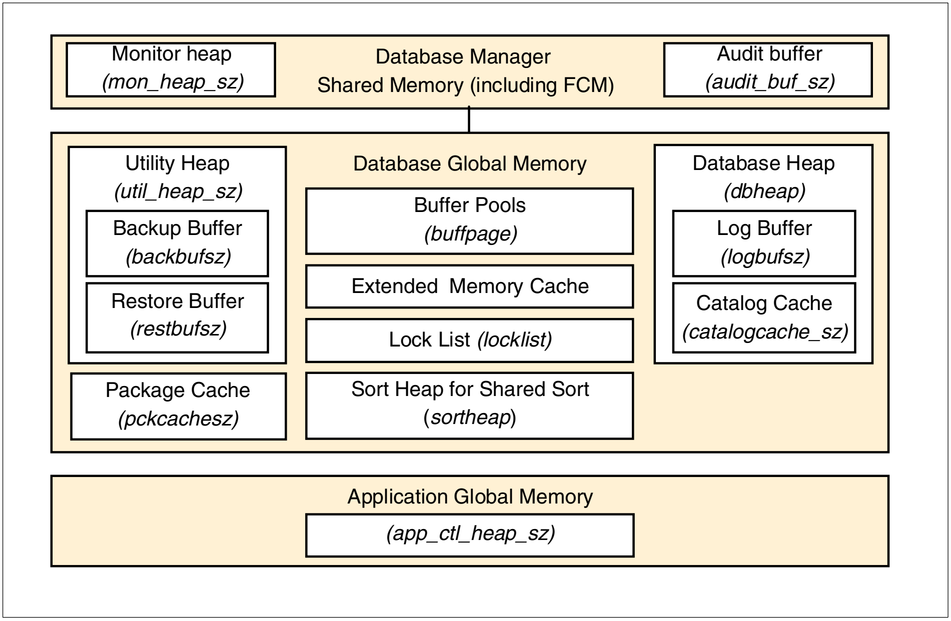


Figure 39. Database manager shared memory overview

The Database Manager Shared Memory is required for the database manager to run. The size of this memory is affected by the following configuration parameters:

- Database System Monitor Heap size (MON_HEAP_SZ)
- Audit Buffer Size (AUDIT_BUF_SZ)
- FCM Buffers (FCM_NUM_BUFFERS)
- FCM Message Anchors (FCM_NUM_ANCHORS)
- FCM Connection Entries (FCM_NUM_CONNECT)
- FCM Request Blocks (FCM_NUM_RQB)

The database manager uses the fast communication manager (FCM) component to transfer data between DB2 agents when intra-partition parallelism is enabled. Thus if you do not enable intra-partition parallelism, memory areas required for FCM buffers, message anchors, connection entries and request blocks are not allocated. We discuss these parameters later in this chapter.

The maximum size of the Database Global Memory segment is determined by the following configuration parameters:

- Buffer Pool Size that were explicitly specified when the buffer pools were created or altered (the value of BUFFPAGE database configuration parameter is taken if -1 is specified)
- Maximum Storage for Lock List (LOCKLIST) (see Chapter 7)
- Database Heap (DBHEAP)
- Utility Heap Size (UTIL_HEAP_SZ) (see Chapter 8)
- Extended Storage Memory Segment Size (ESTORE_SEG_SZ) (see Chapter 3)
- Number of Extended Storage Memory Segments (NUM_ESTORE_SEGS) (see Chapter 3)
- Package Cache Size (PCKCACHESZ)

Application Global Memory is determined by the following configuration parameter:

- Application Control Heap Size (APP_CTL_HEAP_SZ)

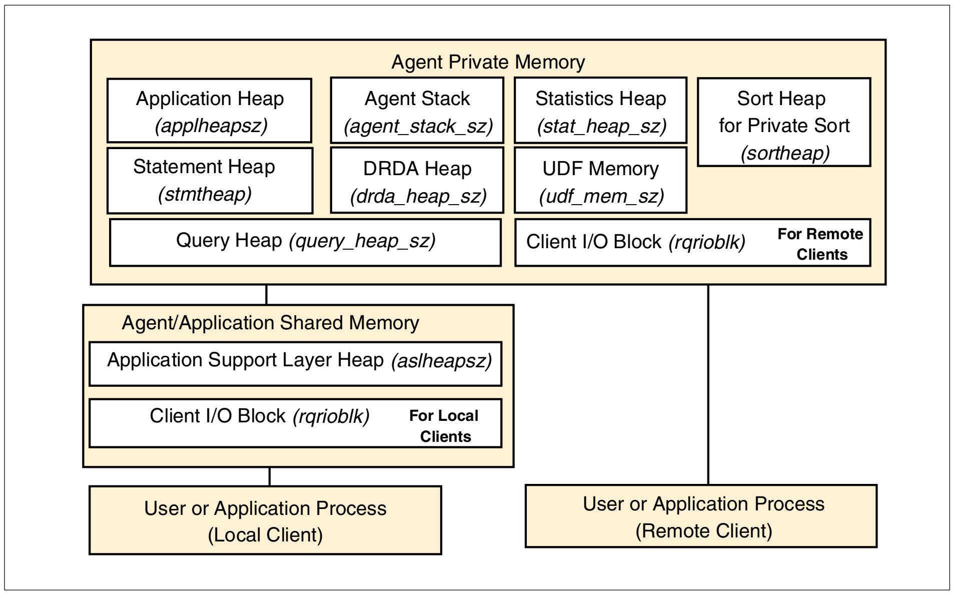

Agent Private Memory
Application Heap (applheapsz)
Agent Stack (agent_stack_sz)
Statistics Heap (stat_heap_sz)
Sort Heap for Private Sort (sortheap)
Statement Heap (stmtheap)
DRDA Heap (drda_heap_sz)
UDF Memory (udf_mem_sz)
Query Heap (query_heap_sz)
Client I/O Block (rqrioblk)
For Remote Clients
Agent/Application Shared Memory
Application Support Layer Heap (aslheapsz)
Client I/O Block (rqrioblk)
For Local Clients
User or Application Process (Local Client)
User or Application Process (Remote Client)

Figure 40. Database agent/application private/shared memory overview

The maximum size of Agent Private Memory segments is determined by the values of the following parameters:

- Application Heap Size (APPLHEAPSZ)
- Sort Heap Size (SORTHEAP)

- Statement Heap Size (STMTHEAP)
- Statistics Heap Size (STAT_HEAP_SZ)
- Query Heap Size (QUERY_HEAP_SZ)
- DRDA Heap Size (DRDA_HEAP_SZ)
- UDF Shared Memory Set Size (UDF_MEM_SZ)
- Agent Stack Size (AGENT_STACK_SZ)
- Client I/O Block Size (RQRIOBLK) (for remote clients)

The size of Agent/Application Shared Memory is affected by the following:

- Application Support Layer Heap Size (ASLHEAPSZ)
- Client I/O Block Size (RQRIOBLK) (for local clients)

For valid ranges and default values of these configuration parameters, see the *DB2 UDB Administration Guide - Performance*, SC09-2945.

6.3 CPU related parameters

Here we discuss the configuration parameters which affect CPU usage. As you have seen in Chapter 1, “Overview” on page 1, a database manager has several DB2 processes, and each of them consumes CPU time. Among the processes, the major consumers of CPU time are DB2 agents, including coordinator agents and subagents. They are one of the most crucial processes and facilitate the operations of applications with databases. If your database server has multiple CPUs and you enable intra-partition parallelism, the database manager can exploit those CPUs using multiple subagents to process one complex query.

6.3.1 Intra-partition parallelism

The database manager configuration parameter INTRA_PARALLEL controls whether or not intra-partition parallelism is enabled. We suggest that this parameter should be enabled or disabled depending on these considerations:

- Typical workload
- The number of users

For complex SQL type workloads with relatively few users (such as OLAP or DSS), then enable intra-partition parallelism (set to YES). For SQL which is simple, repetitive and the number of queries are large (such as OLTP), then do not enable intra-partition parallelism (set to NO).

If you do disable parallelism by setting the INTRA_PARALLEL database manager configuration parameter to NO, we advise that you set the MAX_QUERYDEGREE database manager configuration parameter or DEFAULT_DEGREE database configuration parameters to 1.

With intra-partition parallelism enabled, DB2 will allocate approximately 12 MB of memory (used for control information). This control area will be checked for each statement. So by setting MAX_QUERYDEGREE or DFT_DEGREE to try to avoid parallelism is meaningless as you still incur this overhead. This overhead will not be incurred with INTRA_PARALLEL set to NO.

Here are the additional considerations:

- MAX_QUERYDEGREE and DFT_DEGREE
  Set this database manager configuration parameter to –1 (ANY) if you have set INTRA_PARALLEL to YES. By doing this, you allow the DB2 optimizer to decide the most suitable value. The value it chooses is based on complexity of query, database and instance configuration, and workload.
- CPUSPEED
  Set this database manager configuration parameter to –1. This will be the most recommended value for DB2. By doing this, DB2 will calculate the value.
- For new instances, make your decision on intra-partition parallelism before you create your database(s). If you change this value once a database exists, then you will need to rebind all packages defined in that database.

6.3.2 Controlling the number of DB2 agent processes

Enabling intra-partition parallelism and having more concurrent users will increase the number of DB2 agent processes. For example, if you disable intra-partition parallelism, five concurrent query requests will be carried out by five DB2 agent processes respectively; however, if you enable intra-partition parallelism and the chosen query degree is four, twenty-five DB2 agent processes (one coordinator agents and four subagents for each request) will carry out these five query requests.

Having a huge number of DB2 agent processes may cause CPU constraints due to context switching, as well as memory constraints.

To control the number of DB2 agents, you have the following configuration parameters:

- The database manager configuration parameter MAXAGENTS defines the maximum number of database manager agents, whether it is coordinator agents or subagents, available at any given time to accept application requests.

- The database manager configuration parameter MAX_COORDAGENTS defines the maximum number of coordinator agents.

6.4 Memory related parameters

Many of the configuration parameters available in DB2 affect memory usage on the system. We discuss some of them which have high impact on the database performance.

6.4.1 Sorting methods

When an SQL query requires the data to be returned in a defined sequence or order, the result may or may not require sorting. DB2 will attempt to perform the ordering through index usage. If an index cannot be used, the sort will occur. A sort involves two steps:

1. A sort phase.
2. Return of the results of the sort phase.

How the sort is handled within these two steps results in different categories or types by which we can describe the sort. When considering the sort phase, the sort can be categorized as *overflowed* or *non-overflowed*. When considering the return of the results of the sort phase, the sort can be categorized as *piped* or *non-piped*.

6.4.1.1 Overflowed and non-overflowed

If the information being sorted cannot fit entirely into the sort heap (a block of memory that is allocated each time a sort is performed), it overflows into temporary database tables. Sorts that do not overflow (non-overflowed) always perform better than those that do.

6.4.1.2 Piped and non-piped

If sorted information can return directly without requiring a temporary table to store a final, sorted list of data, it is referred to as a piped sort. If the sorted information requires a temporary table to be returned, it is referred to as a non-piped sort. A piped sort always performs better than a non-piped sort.

The DB2 optimizer will determine if a non-overflowed sort can be performed and if a piped sort can be performed by comparing the expected result set with the value of the SORTHEAP database configuration parameter, and so forth.

Note

To obtain an ordered result set, a sort is not always required. If an index scan is the access method used, then the data is already in the order of the index, and sorting is not required.

The sort heap is used for each application sort request. The DB2 optimizer may decide to attempt to perform a non-overflowed piped sort. However, if there is not enough available memory to allocate another sort heap at run time, then the sort may be performed using a temporary table.

There is a database manager parameter that is used to control the total amount of memory allocated for sorting on the DB2 server. The parameter is called SHEAPTHRES. The SHEAPTHRES parameter should be set to the maximum amount of memory for all sort heaps that should be allowed to be allocated at any given time.

If intra-partition parallelism is enabled, a sort operations can be processed in parallel, and it can be a private sort or a shared sort, which uses memory from two different memory sources. As you can see in Figure 39, the sort heap for a private sort is allocated in the Agent Private Memory; whereas the sort heap for a shared sort is allocated in the Database Global Memory. The size of the shared sort memory area is statically predetermined (and not pre-allocated) at the time of the first connection to a database based on the value of SHEAPTHRES. The size of the private sort memory area is unrestricted.

Note

You can use the Explain Facility to see which type of parallel sort is performed for a query.

The SHEAPTHRES parameter is used differently for private and shared sorts.

For private sorts, this parameter is an instance-wide soft limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private-sort memory consumption for an instance reaches this limit, the memory allocated for additional incoming private-sort requests will be considerably reduced.

For shared sorts, this parameter is a database-wide hard limit on the total amount of memory consumed by shared sorts at any given time. When this limit is reached, no further shared-sort memory requests will be allowed (until the total shared-sort memory consumption falls below the limit specified by `SHEAPTHRES`).

There are a number of sorting related performance elements that can be monitored using the system performance monitor. These elements include:

- **Total sorts** — This variable can be monitored to see the total number of sorts that have been executed.
- **Sort overflows** — This variable can be monitored to see the total number of sorts that ran out of sort heap and may have required disk space for temporary storage.

You can calculate the percentage of overflow sorts by `Sort Overflows/Total Sorts` and use the value to determine if the optimizer is attempting to use a sort heap and fails. If the percentage is high, consider increasing the `SORTHEAP` and/or `SHEAPTHRES` values.

You can also monitor the following performance elements:

- **Piped sorts requested** — This variable can be monitored to see the number of piped sorts that have been requested.
- **Piped sorts accepted** — This variable can be monitored to see the number of piped sorts that have been accepted.

You can calculate the percentage of piped sorts accepted by `Piped Sorts Accepted/Piped Sorts Requested` and determine if piped sorts are being chosen by the optimizer but not accepted. If the percentage is low, consider increasing the `SORTHEAP` and/or `SHEAPTHRES`.

Note

In a piped sort, the sort heap does not get freed until the application closes the cursor associated with the sort. So a piped sort can use up memory until the cursor is closed.

While modifying `SORTHEAP` / `SHEAPTHRES`, you should be aware of the possible implications of either of these options.

- If you increase the sort heap threshold, then there is the possibility that more memory will remain allocated for sorting. This could cause the paging of memory to disk.

- If you decrease the sort heap, more chances for the number of records inserted into a sort exceeds the capacity of the sort heap, and one or more “sort runs” will be written to a single temporary table. Each run constitutes a sorted list of records that must be merged with other runs in order to produce a final sorted list. Thus, you might require an extra external merge to get the final sorted list, that could slow down the overall sort performance.

It is important to allocate as much memory as possible to the sort heap (and set the threshold accordingly) without over allocating memory and causing memory paging to occur. It is possible for a sort to be done entirely in sort memory. However, if this causes operating system to perform page swapping then you can lose the advantage of a large sort heap. So, whenever you adjust the SORTHEAP/SHEAPTHRES configuration parameters, use an operating system monitor to track any changes in the system paging.

The following two sample outputs show monitored sorting related elements by the snapshot monitor:

```
$db2 GET SNAPSHOT FOR DATABASE MANAGER | grep -i sort
Sort heap allocated                          = 305
Post threshold sorts                         = 0
Piped sorts requested                        = 4
Piped sorts accepted                         = 4
Sorting Information  (SORT) = ON  06-21-2000 15:47:36.337746
```

```
$db2 GET SNAPSHOT FOR DATABASE ON dbname | grep -i sort
Total sort heap allocated               = 305
Total sorts                             = 2
Total sort time (ms)                    = 19649
Sort overflows                          = 0
Active sorts                            = 4
```

6.4.2 Agent pool size

The database manager configuration parameter NUM_POOLAGENTS defines the size of the agent pool which contains idle DB2 agents. When DB2 agents finish executing their current request, they will be in idle state unless the number of idle agents exceed the value of NUM_POOLAGENTS; otherwise, they will be terminated.

When the database manager needs to assign a coordinator agent or subagents for a application, it tries to reuse idle agents from agent pool. If the agent pool does not have available idle agents, the database manager will create new agent processes (up to the number defined by MAXAGENTS). If more agents are created than is indicated by the value of NUM_POOLAGENTS, they will be terminated when they finish executing their current request, rather than be returned to the pool.

Setting the appropriate value for this parameter can reduce the cost to create and terminate DB2 agent processes. Too high a value for this parameter may waste the memory due to many idle agents.

If you run a decision-support environment in which few applications connect concurrently, set NUM_POOLAGENTS to a small value to avoid having an agent pool that is full of idle agents.

If you run a transaction-processing environment in which many applications are concurrently connected, increase the value of NUM_POOLAGENTS to avoid the costs associated with the frequent creation and termination of agents.

6.4.3 Disclaim memory areas for DB2 agents

When DB2 agents finish executing their current request and are returned to the agent pool, they do not release their agent private memory which includes the allocated sort heap. This behavior is usually results in good performance, as the memory is kept for fast re-use. However, if you want to increase the agent pool size on a memory constrained system, this behavior may cause excessive activity to the paging space because many idle agents may keep large amount of memory. To avoid this condition, set the DB2 registry variable DB2MEMEDISCLAIM to YES by executing the following command:

```
db2set DB2MEMDISCLAIM = yes
```

Setting DB2MEMDISCLAIM to YES tells DB2 to disclaim some or all memory associated with the agent which is returned to the agent pool, depending on the value given with the DB2 registry variable DB2MEMMAXFREE. This DB2MEMMAXFREE value specifies the amount of memory that can be retained by each DB2 agent. If DB2MEMMAXFREE is null, then all of the memory is disclaimed when the agent is returned to the agent pool. If DB2MEMMAXFREE is given a value, then only some of the memory is kept (up to the value given in DB2MEMMAXFREE). This ensures that the memory is made readily available for other processes as soon as the agent is returned to the agent pool. DB2MEMDISCLAIM and DB2MEMMAXFREE work together. We recommend that if you use this feature, you specify a value of 8 MB for DB2MEMMAXFREE, as the following example shows:

```
db2set DB2MEMMAXFREE = 8000000
```

6.4.4 FCM related parameters

If you enable intra-partition parallelism, the database manager will use FCM component to transfer data between DB2 agents. Therefore, the amount of memory area defined by the following database manager configuration parameters would be allocated in the Database Manager Shared Memory even though you do not use DB2 EEE, which enables you to create partitioned databases:

- FCM Buffers (FCM_NUM_BUFFERS)
- FCM Message Anchors (FCM_NUM_ANCHORS)
- FCM Connection Entries (FCM_NUM_CONNECT)
- FCM Request Blocks (FCM_NUM_RQB)

You can start with the values recommended by Configure Performance Wizard. To tune these parameters, use the database system monitor to monitor the low water mark for the free buffers, free message anchors, free connection entries, and the free request blocks. If the low water mark is less than 10 percent of the number of the corresponding free data item, increase the value of the corresponding parameter. We discuss the database system monitor in Chapter 5, “Monitoring tools and utilities” on page 119.

6.4.5 Package cache size

The PCKCACHESZ database configuration parameter defines the package cache size. The database manager uses this memory to cache packages, which has been loaded from the system catalog, for static SQL or (dynamically generated) for dynamic SQL. If applications connecting to a database execute the same query multiple times, the database manager can reduce its internal overhead by eliminating the need to reload sections of package for static SQL, and also can reduce overhead to generate the package for dynamic SQL.

For dynamic SQL, even though your application does not execute exactly the same queries, it may benefit by the package cache using parameter markers. See 7.3.2.2, “Avoid repeated PREPARE statements” on page 246.

There are a number of package cache related performance variables that can be monitored using the system performance monitor. These parameters include:

- **Package cache insert** — This variable indicates the total number of times that a requested section was not available for use and had to be loaded into the package cache. If this number is high, it may mean that the cache is too small.

- **Package cache lookups** — This variable indicates the number of times that an application looked for a section or package in the package cache. This counter includes the cases where the section is already loaded in the cache and when the section has to be loaded into the cache.

You can calculate the package cache hit ratio using the following formula:

```
1 - (Package Cache Inserts / Package Cache Lookups)
```

Normally, a low hit ratio would mean that the cache is too small. But as a result of running DDL statements, a low hit ratio does not always mean that the cache is small. It may be caused by invalidated dynamic sections and the need of information to be reinserted back into the cache. In this case, increasing PCKCACHESZ will not improve the cache performance.

You should take your application type into account when you tune the package cache size. For example, if your applications rarely execute the same query, increasing the package cache to keep the package may not be worthwhile to do.

The package cache is a working cache, so you cannot set this parameter to zero. There must be sufficient memory allocated in this cache to hold all sections of the SQL statements currently being executed. If there is more space allocated than currently needed, then sections are cached. These sections can simply be executed the next time they are needed without having to load or compile them.

The limit specified by the PCKCACHESZ parameter is a soft limit. This limit may be exceeded, if required, if memory is still available in the database shared set. You can use the Package Cache High Water Mark monitor element to determine the largest that the package cache has grown, and the Package Cache Overflows monitor element to determine how many times the limit specified by the PCKCACHESZ parameter has been exceeded. The following example shows monitored package cache related elements by the snapshot monitor:

```
$db2 GET SNAPSHOT FOR DATABASE ON dbname | grep -i package
Package cache lookups                    = 28
Package cache inserts                    = 17
Package cache overflows                  = 0
Package cache high water mark (Bytes)    = 221566
```

6.4.6 Utility heap size

The `UTIL_HEAP_SZ` database configuration parameter defines the maximum size of the utility heap which is allocated in the Database Global Memory. The DB2 utilities including `BACKUP`, `RESTORE`, `LOAD use` this memory area. Each database has a utility heap allocated as required by the utilities, and freed when the utilities are completed and the memory is no longer needed. We recommend to use the default value for this parameter unless your utilities run out of space. See Utility Heap Size in 8.1.2.1, “Utility heap size (util_heap_sz)” on page 274.

6.5 Disk I/O related parameters

Hardware components that make up your system can influence the performance of your system. You have to balance the demand for disk I/O across several disk storage devices, controllers. These can affect how fast the database manager can retrieve information from disks. The appropriate values for configuration parameters can be determined by benchmarking, where typical and worst-case SQL statements are run against the server and the values of the parameters are modified until the point of diminishing return for performance is found.

Parameters that allocate memory should never be set to their highest values, unless it is fully justified, even on systems with the maximum amount of memory installed. A few tuning parameters that have implications associated with performance of I/O are discussed here.

On OLTP (that is, random access) environments, you need as many disks in the table spaces as are necessary to support the I/O rates required. For example, if your table space needs to process 100 IO/sec and your disks can service 60 IO/sec, then you would need two disks. Often writes are more expensive than reads (as with RAID systems, since they may require multiple physical writes for redundancy), and thus the read/write ratio, along with the true I/O cost of one write, has to be considered.

On DSS (that is, sequential access) environments, you need as many disks as are necessary to meet the I/O rates required. Note, however, that DSS workloads are more “disk-friendly”, mainly due to their larger I/O sizes. Because of this, and the prefetching/caching abilities built into many file systems and disk subsystems (for example, RAID/Extended Storage Subsystem), a DSS workload can often get by with less disks than an OLTP system with similar I/O needs. Normally, 5 disks per CPU are often sufficient for simple DSS scans. With full SMP parallelism enabled, this number can increase to 10 or more disks per CPU.

6.5.1 Buffer pool size (buffpage)

A buffer pool is an area of storage in memory into which database pages are temporarily read and changed. The purpose of the buffer pool is to improve database system performance by buffering the data in memory. Here data can be accessed from memory rather than from disk, so the database manager needs to read or write less to the disk. Not all data in DB2 is buffered; long field and LOBs are only accessed through direct I/O and are never stored in the buffer pool.

One component of the database manager is called Buffer Pool Services (BPS). The BPS is responsible for reading data and index pages from disk into memory and writing pages from memory to disk. BPS will use the File Storage Manager or the Raw Storage Manager to get the pages depending on whether an SMS or DMS table space is used.

When creating a buffer pool, by default, the page size is 4 KB, but you can choose to have the page size set at one of these values: 4 KB, 8KB, 16 KB, or 32 KB.
If buffer pools are created using one page size, then only table spaces created using the identical page size can be associated with them. You cannot alter the page size of the buffer pool following its creation.

Each database has at least one buffer pool (IBMDEFAULTBP, which is created when the database is created), and you can have more also. All buffer pools reside in the Database Global Memory (as shown in Fig. 39 on page 197), which is available to all applications using the database. All buffer pools are allocated when the first application connects to the database, or when the database is explicitly activated using the ACTIVATE DATABASE command. Use this ACTIVATE DATABASE command to keep buffer pool primed even if all the connections terminate. This will be very useful when connection load is highly dynamic (for example, Web servers).

As an application requests data out of the database, pages containing the data are transferred to one of the buffer pools from disk. The next time an application requests data, the buffer pool is checked first to see if the data is there in the memory area; if it is found, BPS does not need to read data from the disk. Avoiding data retrieval from disk storage results in faster performance. If the buffer pools are not large enough to keep the required data in memory, the BPS has to read data from disk. Pages are not written back to the disk until the page is changed, or one of the following occurs:

- All applications disconnect from the database.
- The database is explicitly deactivated.

- The database quiesces (that is, all connected applications have committed)
- Space is required for another incoming page
- A page cleaner is available (NUM_IOCLEANERS database configuration parameter is not zero) and is activated by the database manager.

Note

After changed pages are written out to disk, they are not removed from the buffer pool unless the space they occupy is needed for other pages.

The BUFFPAGE database configuration parameter controls the size of a buffer pool when the CREATE BUFFERPOOL or ALTER BUFFERPOOL statement was run with NPAGES -1; otherwise, the BUFFPAGE parameter is ignored and the buffer pool will be created with the number of pages specified by the NPAGES parameter. Thus, each buffer pool that has a NPAGES value of -1 uses BUFFPAGE.

Note

NPAGES in SYSCAT.BUFFERPOOLS overrides BUFFPAGE.

To determine whether the BUFFPAGE parameter is active for a buffer pool, issue this command:

```
SELECT * FROM SYSCAT.BUFFERPOOLS
```

Instead of using the BUFFPAGE configuration parameter, you can use the CREATE BUFFERPOOL and ALTER BUFFERPOOL SQL statements to create and change buffer pools and their sizes. See *SQL Reference*, SC09-2974 for the syntax.

6.5.1.1 Buffer pool hit ratio and index pool hit ratio

If you create your own buffer pools in addition to the default buffer pool, you must be careful how you allocate space for each one. Allocating a large buffer pool to a table space containing a large number of small, rarely used tables and a small buffer pool to a tables space containing a large, frequently accessed table will lead to performance problems. The size of the buffer pools should reflect the size of table in the table space, and how frequently they are updated or queried.

The DB2 optimizer will utilize the different buffer pools to achieve the best query performance. When selecting the access plan, the optimizer considers the I/O cost of fetching pages from disk to the buffer pool. In its calculations, the optimizer will estimate the number of I/Os required to satisfy a query. This estimate includes a prediction of buffer pool usage, since additional physical I/Os are not required to read rows in a page that is already in the buffer pool. The optimizer considers the value of the `NPAGES` column in the `BUFFERPOOLS` system catalog table in estimating whether a page will be found in the buffer pool.

The I/O costs of reading the tables can have an impact on:

- How two tables are joined, for example, outer versus inner.
- Whether an index will be used to read the data.

`AVG_APPLS` parameter provides the optimizer with information regarding the average number of active applications. This parameter is used by the optimizer to determine how much of each buffer pool may be used for each application. A value of 1 for this parameter will cause the optimizer to treat the entire buffer pool is available to one application.

Snapshot monitor for buffer pools can be used to capture information on the number of reads and writes and the amount of time taken. Before getting the snapshot, set the buffer pool monitor switch on by performing either of the following actions:

- Execute `UPDATE MONITOR SWITCHES` command which affect only the current session.
- Update the `DBF_MON_BUFPOOL` database manager configuration parameter to `ON`.

Snapshot monitor will provide information about buffer pool activity for all active databases when you run the following command:

```
GET SNAPSHOT FOR ALL BUFFERPOOLS
```

The following example shows a snapshot for buffer pools:

```
 Bufferpool Snapshot

Bufferpool name                          = TPCDDATABP
Database name                            = TPC
Database path                            = /database/db2inst1/NODE0000/SQL0000
1/
Input database alias                     =
Buffer pool data logical reads           = 4473
Buffer pool data physical reads          = 207
Buffer pool data writes                  = 20
Buffer pool index logical reads          = 0
Buffer pool index physical reads         = 0
Total buffer pool read time (ms)         = 218
Total buffer pool write time (ms)        = 0
Asynchronous pool data page reads        = 200
Asynchronous pool data page writes       = 20
Buffer pool index writes                 = 0
Asynchronous pool index page reads       = 0
Asynchronous pool index page writes      = 0
Total elapsed asynchronous read time     = 131
Total elapsed asynchronous write time    = 0
Asynchronous read requests               = 7
Direct reads                             = 0
Direct writes                            = 0
Direct read requests                     = 0
Direct write requests                    = 0
Direct reads elapsed time (ms)           = 0
Direct write elapsed time (ms)           = 0
Database files closed                    = 0
Data pages copied to extended storage    = 0
Index pages copied to extended storage   = 0
Data pages copied from extended storage  = 0
Index pages copied from extended storage = 0
```

If the server needs to read a page of data, and that page is already in the buffer pool, then it will be accessed faster than the one from the disk. It is then desirable to “hit” as many pages as possible in the buffer pool.

The following data elements can be measured to evaluate how the buffer pool is being used:

- Buffer Pool Data Logical Reads: Denotes the total number of read data requests that went through the buffer pool.
- Buffer Pool Data Physical Reads: Denotes the number of read requests performed that required I/O to place data pages in the buffer pool.
- Buffer Pool Index Logical Reads: Denotes the total number of read requests for index pages that went through the buffer pool.

- Buffer Pool Index Physical Reads: Denotes the number of read requests for index pages that require I/O activity to place index pages in the buffer pool.

The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk in order to service a page request. That is, the page was already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk I/O.

The overall buffer pool hit ratio can be calculated as the difference between the number of all (data + index) logical reads and number of all (data + index) physical reads divided by the total number of read requests.

$$BufferPoolHitRatio = \frac{\Sigma LogicalReads - \Sigma PhysicalReads}{\Sigma LogicalReads} \times 100$$

Similarly, an index pool hit ratio is calculated as the difference between the number of the index logical reads and the number of index physical reads divided by the total number of index read requests.

$$IndexPoolHitRatio = \frac{\Sigma IndexLogicalReads - \Sigma IndexPhysicalReads}{\Sigma IndexLogicalReads} \times 100$$

Increasing buffer pool size will generally improve the hit ratio, but you will reach a point of diminishing returns. Ideally, if you could allocate a buffer pool large enough to store your entire database, then once the system is up and running, you would get a hit ratio of 100%. However, this is unrealistic in most cases. The significance of the hit ratio depends on the size of your data, and the way it is accessed.

For a large database, increasing the buffer pool size may have minimal effect on the buffer pool hit ratio. Its number of data pages may be so large, that the statistical chances of a hit are not improved increasing its size. But you might find that tuning the index buffer hit ratio achieves the desired result. This can be achieved using two methods:

1. Split the data and indexes into two different buffer pools and tune them separately.
2. Use one buffer pool, but increase its size until the index hit ratio stops increasing.

The first method is often more effective, but because it requires indexes and data to reside in different table spaces, it may not be an option for existing databases. It also requires you to tune two buffer pools instead of one, which can be a more difficult task, particularly when memory is constrained.

A very large database where data is accessed evenly would have a poor hit ratio. There is little you can do with very large tables. In such a case, you would focus your attention on smaller, frequently accessed tables, and on the indexes — perhaps assigning them to individual buffer pools, in which you can aim for higher hit ratios.

The index pool hit ratio and the buffer pool hit ratio are influenced by data reorganization. it is always advisable to perform a `REORGCHK` command to check these hit ratios before tuning.

6.5.1.2 Prefetching

Prefetching means retrieving one or more pages from disk in anticipation of their use. Prefetching index and data pages into the buffer pool can help improve performance by reducing the time spent waiting for I/O to complete. Setting the prefetch size has significant performance implications, particularly for large table scans.

Prefetchers are used to bring data into the buffer pool and also look after dropping temporary tables. Page cleaners are used to move data from the buffer pool back out to disk. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk to buffer pool, if there were no data prefetchers. Agents of the application send asynchronous read-ahead requests to a common prefetch queue. When prefetchers are available, they perform these requests by using big-block or scatter read operations to bring the requested pages from disk to the buffer pool.

To enable prefetching, the database manager starts separate threads of control, known as I/O servers, to perform page reading. As a result, the query processing can be separated into two parallel activities: data processing (CPU) and data page I/O. The I/O servers wait for prefetch request from the CPU processing activity. No more than this number of I/Os for prefetching and utilities can be in progress for a database at any time. Non-prefetch I/Os are scheduled directly from the database agents.

If the prefetchers and page cleaners are not specified, then the application agents would have to do all the reading and writing of data between the buffer pool and disk storage.

A good value, in order to fully exploit all the I/O devices in the system, is 1 or 2 more than the number of physical devices on which the database resides. Since any unused I/O servers will remain idle and associated overhead with each I/O server is minimum, it is always better to have additional I/O servers.

The number of prefetchers (`NUM_IOSERVERS`) is set to at least the number of physical disks in the database to maximize the opportunity for parallel I/O. Because one I/O server can serve only one I/O device (disk), it is recommended to configure one or more `NUM_IOSERVERS` than the number of physical devices on which the table space container reside. It is better to use additional I/O servers because of its minimal overhead.

Efficient I/O can be obtained in DSS, since prefetchers provide asynchronous access, big blocks, and use parallel I/Os. Thus, the agents seldom need to wait on the flush of a dirty page.

In OLTP environments, page cleaners provide background buffer pool page cleaning, so that the agents rarely need to wait on the flush of a dirty page.

6.5.1.3 Page cleaners

Pages in the buffer pool can have different attributes:

- “In-use” pages are currently being read or updated.
- “Dirty” pages are waiting to be “cleaned”.

Dirty pages are pages where data has been changed but has not yet been written to disk. After a page is written to disk, it is considered clean, and remains in the buffer pool.Page cleaners check the buffer pool, and write pages asynchronously to disk. Page cleaners basically perform two operations:

- Assure that an agent will always find free pages in the buffer pool.
- Copy the oldest unwritten change of Logical Sequence Number (LSN) to the buffer pool.

Without the existence of the independent page cleaners, the DB2 agents have to clean dirty pages by themselves, and the application being served by these agents ends up with a poor response.

Pages are written from buffer pool to disk when the percentage of space occupied by changed pages in the buffer pool has exceeded the value specified in the `CHNGPGS_THRESH` configuration parameter. Page cleaner agents (`NUM_IOCLEANERS`) write out changed pages to disk. Page cleaners are sometimes referred to as asynchronous page cleaners or asynchronous buffer writes because they are independent of DB2 agents.

The NUM_IOCLEANERS parameter allows you to specify the number of asynchronous page cleaners for a database. These page cleaners write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. This means that the agents will not wait for changed pages to be written out. As a result, your application's transaction can run faster.

You may use the changed pages threshold (CHNGPGS_THRESH) to specify the level (in percentage) of changed pages at which the asynchronous page cleaners will be activated, if they are not currently active. When the page cleaners are started, they will build a list of pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start. For the databases with a heavy update transaction workload, setting CHNGPG_THRESH to default (60) is recommended. A percentage larger than the default can help performance if your database has a small number of very large tables.

If the NUM_IOCLEANERS parameter is zero (0), no page cleaners are started and, as a result, database agents will perform all the page writes from the buffer pool to disk. If you have many physical storage devices, this leads to a significant performance impact, and there is a greater chance that one or more devices will be idle.

6.5.1.4 Tuning disk I/O: summary

When tuning the buffer pool size, the goal is to reduce the number of I/O operations by placing the data pages in the buffer pool before they are needed. Applications will wait on synchronous I/Os, so buffer pool operations performed by I/O cleaners (writing pages from the buffer pool to disk) should be performed asynchronously whenever possible. The configuration parameters involved are:

- BUFFPAGE
- PREFETCHSIZE
- NUM_IOCLEANERS
- CHNGPGS_THRESH
- NUM_IOSERVERS

To detect if the buffer pool is undersized, the best way is to:

- Check the number of times the page cleaners were triggered.
- Check I/O servers not prefetching as many pages as expected.

The first approach, used when tuning the buffer pool size, is to increase the buffer pool until you get a good index pool hit ratio (around 80%). Then perform tuning on other configuration parameters: NUM_IOCLEANERS, CHNGPGS_THRESH, NUM_IOSERVERS and PREFETCHSIZE.

A good value for the number of page cleaner (NUM_IOCLEANERS), in an OLTP environment, is the number of physical devices/disks used by the database +2. The default is 1. Use larger values for update-intensive workloads, or when the number of data or index page writes is large with respect to the number of async data or index page writes. In a DSS environment, the number of CPU is a good value for NUM_IOCLEANERS. To monitor the I/O cleaners activity, execute the following command:

```
db2 get snapshot for database on <dbname> | grep writes
```

The output would be like the following:

```
Buffer pool data writes                    = 0
Asynchronous pool data page writes         = 0
Buffer pool index writes                   = 0
Asynchronous pool index page writes        = 0
Direct writes                              = 72
```

A good value for CHNGPGS_THRESH is to use default value (60%), is suitable for typical workloads. A percentage larger than the default can help performance if your database has a small number of very large tables.

The number of prefetchers (NUM_IOSERVERS) is set to at least the number of physical disks in the database to maximize the opportunity for parallel I/O. Because one I/O server can serve only one I/O device (disk), it is recommended to configure one or more NUM_IOSERVERS than the number of physical devices on which the table space container reside. It is better to use additional I/O servers because of its minimal overhead.

When sequential I/O is important (for example, DSS workloads) ensure DMS table spaces are set up well for prefetching and parallel I/O:

1. Multiple containers, ideally at least 5
2. Each container mapped to its own disk(s)
3. Set PREFETCHSIZE to at least (#containers * EXTENTSIZE)

For most purposes, PREFETCHSIZE should be set to #containers * EXTENTSIZE. If you want more aggressive prefetching you can double this, or reduce it if you want to have a less aggressive prefetching, but it is recommended to have a multiple of EXTENTSIZE. Note that over-prefetching wastes buffer pool space, and that under-prefetching leads to prefetch wait time, or the agent ends up doing the I/O.

If you do increase the PREFETCHSIZE beyond the above suggestion, then ensure that the NUM_IOSERVERS database configuration parameter is at least PREFETCHSIZE/EXTENTSIZE. To disable prefetching on a table space, you can set PREFETCHSIZE to zero.

6.5.2 Extended STORagE (ESTORE)

When your machine has more real addressable memory than the maximum amount of virtual addressable memory (between 2 GB to 4 GB on most platforms), then you can configure additional real addressable memory beyond virtual addressable memory as an extended storage cache. Such an extended storage cache can be used by any of the defined buffer pools and should improve the performance of the database manager. The extended storage cache is defined in terms of memory segments.

ESTORE is a second level page cache, and the buffer pool is the primary cache. Because an extended storage cache is an extension to a buffer pool, it must always be associated with one or more specific buffer pools. Each buffer pool can be configured to use it or not. ESTORE is used when the amount of memory that is allowed to be mapped by a single process is less than the total amount of memory available. For UNIX platforms, this extra memory can be accessed by cycling many segments through one address.

Copying pages to and from ESTORE costs CPU (for both the copies and attaches), but it saves I/O every time a disk read is avoided. If you are continually copying pages to ESTORE but rarely read from it, or if you are already more CPU-bound than I/O-bound, then ESTORE will not help much in performance.

ESTORE, on AIX, works by allocating number of extended storage memory segments available for use by the database (NUM_ESTORE_SEGS) and the number of pages in each extended memory segments in the database (ESTORE_SEG_SIZE).

On AIX, when the total number of pages allocated to ESTORE is (NUM_ESTORE_SEGS * ESTORE_SEG_SIZE). While allocating the ESTORE, ensure that the NUM_ESTORE_SEGS should be at least two. If the number of segments is one, then you should turn off ESTORE and allocate the pages to the buffer pools, because you can allocate the same amount of memory to buffer pool(s). Also ensure that your ESTORE segment size is less than 256 MB.

To exploit main memories larger than 4 GB on 32-bit systems, you have the following two options:

- Use ESTORE. It is used especially for read-mostly workloads, or workloads involving large temporary tables that would otherwise spill to disk.
- Also, use DB2_MMAP_READ=NO and DB2_MMMAP_WRITE=NO registry settings on AIX and SMS or DMS FILE table space containers. This frees up an extra 256 MB memory segment and enables the use of the AIX JFS file system cache.

See 3.4.3, “Extended storage” on page 76 for additional information, such as when to use and when not to use ESTORE.

6.5.3 Logging

All changes to data pages are logged, and the updated data page is not written to disk storage before its associated log record is written to the log. Therefore, improving the logging performance is very important to improve the overall performance if your application performs frequent updates. The following configuration parameters affect the logging performance.

6.5.3.1 Log buffer size

Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently and more log records will be written at each time. The number of write operations is also higher if there is not enough space in the log buffer.

The log records are written to disk when one of the following occurs:

1. A transaction commits, or a group of transactions commits (MINCOMMIT).
2. The log buffer is full.
3. As a result of some other internal database manager event.

Log Buffer, whose size is determined by LOGBUFSZ database configuration parameter, holds log records in storage until they are written to disk. Ensure that the log buffer should be large enough to hold the amount of space needed to perform a rollback of current uncommitted transactions without having to read data from the log files.

When you set the log buffer size, consider the size of database heap (DBHEAP), since LOGBUFSZ allocated from the database heap (DBHEAP). Also note that using larger log buffers will not reduce data integrity, as data base commits will still force log file writes.

For OLTP workloads, it is recommended to increase the default log buffer (logbufsz).

6.5.3.2 MINCOMMIT database configuration parameter

By default, write operations to log files will be performed every time a transaction is committed. These write operations to log files can be reduced by grouping commits together. This will reduce the number of writes, thus improving the performance of the database, as a compromise with the response time of small transactions, since they have to wait for other transactions to commit their work.

If there are not enough transactions to commit their work, the database manager will commit transactions every second.

When transactions are short, the log I/O can become a bottleneck due to the frequency of the flushing of the log at COMMIT time. In such environments, setting the `MINCOMMIT` configuration parameter to a value greater than 1 can remove the bottleneck. When a value greater than 1 is used, the COMMITs for several transactions are held or batched. The first transaction to COMMIT waits until (`MINCOMMIT - 1`) more transactions COMMIT; and then the log is forced to disk and all transactions respond to their applications. The result is only 1 log I/O instead of several individual log I/Os.

By increasing `MINCOMMIT` and grouping the COMMITs, an increase in efficient file I/O logging will be achieved, as it will occur less frequently and write more log records each time a write is required. In order to avoid an excessive degradation in response time, each transaction only waits up to 1 second for the (`MINCOMMIT`- 1) other transactions to COMMIT. If the 1 second of time expires, the waiting transaction will force the log itself and respond to its application.

Changes to the value specified for this parameter take effect immediately; you do not have to wait until all applications disconnect from the database.

Increase or change in this parameter from its default value when there are multiple or write applications typically request concurrent database COMMITs. This will result in writing more log records each time it occurs. If you increase `MINCOMMIT`, you may also need to increase the `LOGBUFSZ` parameter to avoid having a log-full buffer force a write during these heavy load periods.

For OLTP workloads, it is recommended to increase `MINCOMMIT` to a value of 10 or more.

6.5.3.3 SOFTMAX database configuration parameter

The SOFTMAX database configuration parameter represents the percentage of LOGFILSIZ when a softcheck point will either write the log control file to disk or call an asynchronous page cleaner.

If you use very large log files, consider lowering this parameter, as this will reduce the time required to restart a database in the event of a crash recovery. However, lowering the value can increase the database logging overhead, which may impact performance. Lowering the value is not recommended if you have relatively small log files, as the checkpoint will be reached when these logs become full. See *Administration Guide: Performance*, SC09-2945 for additional information.

6.5.3.4 LOGFILSIZ database configuration parameter

The LOGFILSIZ parameter defines the size of each primary and secondary log file. Increase the LOGFILSIZ database configuration parameter if the database has a large number of update/delete/insert transactions, because these transactions cause the log file to become full very quickly, for example, OLTP workloads. A log file that is too small can affect system performance because of the overhead of archiving old log files, allocating new log files, and waiting for a usable log file.

Setting LOGFILESIZ to an excessively high value has a negative impact, such as:

- Taking a long time to create the log files
- Reducing your flexibility when managing archived log files and copies of log files, since some media may not be able to hold an entire log file

If the disk space is scarce, the value of the LOGFILSIZ should be reduced, since primary logs are preallocated at this size.

6.5.3.5 Audit buffer size

The DB2 Audit Facility generates, and allows you to maintain, an audit trail for a series of predefined database and instance events. The records generated from this facility are kept in an audit log file.

The timing of the writing of auditing records to the audit log file can have a significant effect on the performance of databases in the instance. The writing of audit records can take place synchronously or asynchronously with the occurrence of the events causing the generation by those records.

Setting the database manager configuration parameter `AUDIT_BUF_SZ=0` means that the writing is done synchronously (default). The event generating the audit record will wait until the record is written to disk.

Settings greater than 0 indicate that audit records will be written asynchronously to disk storage and improve performance. The value is the number of 4 KB pages for an internal buffer allocated in the Database Manager Shared Memory. If you choose asynchronous mode, the event generating the audit record will not wait and can continue its operation.

In asynchronous mode, there maybe some records lost if an error occurs when they are written into disk. If asynchronous, since we buffer (audit_bf_sz * 4096) bytes worth of data, if a write error occurs, then you will lose that amount of data. If synchronous, you will lose a single record at the time when the write fails.

For more information about the DB2 Audit Facility, refer to “Auditing DB2 Activities” in the *DB2 UDB Administration Guide: Planning*, SC09-2946.

6.6 Network related parameters

Here we discuss a few configuration parameters that improve communication between the client and the server. These parameters mainly depends on the server’s resource allocation and applications.

6.6.1 Number of TCP/IP connection managers

Prior to the Version 7.1 release, when working in a client-server environment where TCP/IP is used for communication, one connection manager is created, irrespective of the number of processors in the system. Since Version 7.1, more than one connection manager can be created using `DB2TCPCONNMGRS` registry variable. You can create up to a maximum of 8 connection managers. If the value is not set, the default number of connection managers will be created. The default value is 1 on single processor machines. For SMP machines, the default number of the TCP/IP connection managers is calculated using:

```
Square Root(# of processors) rounded up to the maximum value of 8
```

The default value can be overridden when the `DB2TCPCONNMGRS` registry value is set; then the number of TCP/IP connection managers specified (to the maximum of 8) will be created.

If the `DB2TCPCONNMGRS` value is less than 1, then the `DB2TCPCONNMGRS` value is set to 1, and a warning message is logged that the value is out of range.

If the DB2TCPCONNMGRS value is greater than 8, then the DB2TCPCONNMGRS value is set to 8, and a warning message is logged that the value is out of range.

If the DB2TCPCONNMGRS values are between 1 and 8, then they are used as given.

When there is greater than one connection manager created, connection throughput should improve when multiple client connections are received simultaneously.

6.6.2 Blocking

Row blocking is a technique that reduces database manager overhead by retrieving a block of rows in a single operation. These rows are stored in a cache, and each fetch request in the application gets the next row from the cache. When all the rows in a block have been processed, another block of rows is retrieved by the database manager.

The cache is allocated when an application issues an OPEN CURSOR request and is deallocated when the cursor is closed. The size of the cache is determined by a configuration parameter which is used to allocate memory for the I/O block. The parameter used depends on whether the client is local or remote:

- For local applications, the parameter ASLHEAPSZ is used to allocate the cache for row blocking
- For remote applications, the parameter RQRIOBLK (client I/O block size) on the client workstation is used to allocate the cache for row blocking. The cache is allocated on the database client

For local applications, you can use the following formula to estimate how many rows are returned per block:

```
Rows per block = ASLHEAPSZ * 4096 / ORL
```

Here, ASLHEAPSZ is in pages of memory 4096 is the number of bytes per page, and ORL is the output row length in bytes.

For remote applications, you can use the following formula to estimate how many rows are returned per block:

```
Rows per block = RQRIOBLK / ORL
```

Here, RQRIOBLK is in bytes of memory, and ORL is the output row length in bytes.

Note that if you use the FETCH FIRST n ROWS ONLY clause or the OPTIMIZE FOR n ROWS clause in a SELECT statement, the number of rows per block will be the minimum of the following:

- The value calculated in the above formula
- The value of n in the FETCH FIRST clause
- The value of n in the OPTIMIZE FOR clause

Refer to the *SQL Reference*, SC09-2974 for more information about cursors.

6.6.2.1 Application support layer heap size (ASLHEAPSZ)

The application support layer heap size is the communication buffer between the local application and its associated agent. For local applications, the ASLHEAPSZ parameter is used to allocate cache for row blocking. This buffer is allocated as shared memory by each database manager agent that is started.

By default, this value is 15 (4 KB) pages. This value can contain an average request or reply between the application and its agent. Its size can be increased if queries retrieving large amounts of data.

If the request or reply do not fit into the buffer, they will be split into two or more send-and-receive buffers. This size should be set appropriately, so that it is able to handle the majority of requests using a single send-and-receive buffer. The size of the request is based on the storage required to hold:

- The input SQLDA
- All of the associated data in the SQLVARs
- The output SQLDA
- Other fields which do not generally exceed 250 bytes

This parameter is also used to determine the I/O block size when a blocking cursor is opened and allocated in the application's private address space. So, this is an additional parameter to be considered while allocating an optimal amount of private memory for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, then a non-blocking cursor will be opened.

The ASLHEAPSZ parameter is used to determine the initial size of the query heap for both local and remote clients. The maximum size of the query heap is defined by the QUERY_HEAP_SZ parameter. QUERY_HEAP_SZ is the set of contiguous memory allocated by the database manager to receive the data sent by the local application. The query heap is used to store each query in the agent's private memory.

The information for each query consists of:

- Input SQLDA
- Output SQLDA
- The statement text
- SQLCA
- The package name
- Creator
- Session number
- Consistency token

This query heap parameter is provided to ensure that an application does not consume unnecessarily large amount of virtual memory within an agent, and it initially set with the value equal to ASLHEAPSZ.

The formula to calculate the ASLHEAPSZ value in number of pages is as follows:

```
aslheapsz >=( sizeof(input SQLDA) +
              sizeof (each input SQLVAR)+sizeof(output SQLDA)+ 250 )
              / 4096
```

You can reduce the size of the ASLHEAPSZ value when:

- The application requests size is small.
- Application is running on a memory constrained system.

You can increase the size of the ASLHEAPSZ value when:

- The queries are very large and require more than one send and receive request.
- The application is not running on a memory constrained system.

Larger record blocks may also cause more fetch requests than are actually required by the application. Control the fetch requests by using the OPTIMIZE FOR clause on the SELECT statement in your application.

6.6.2.2 Client I/O block size (RQRIOBLK)

This parameter specifies the size of the communication buffer between remote applications and their database agents on the database server. The client I/O block of memory is used to serve the request and replies from and to remote applications. When a database client requests a connection to a remote database, this communication buffer is allocated on the client.

On the database server, a communication buffer of 32767 bytes is initially allocated, until a connection is established and the server can determine the value of RQRIOBLK at the client. Once the server knows the RQRIOBLK value, it will reallocate its communication buffer if the value on the client is not 32767 bytes. It is independent of the protocol used to connect the client and server.

In addition to the communication buffer, the RQRIOBLK parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

You need to increase the value of RQRIOBLK for the following conditions or scenarios:

- A single SQL statement transmits large data (for example, large object data).
- The number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, this will increase the size of the working set memory for each connection, a trade-off has to be done.
- Larger record blocks may also cause more fetch requests than are actually required by the application. Control the fetch requests by using the OPTIMIZE FOR clause on the SELECT statement in your application.

6.6.2.3 DOS requester I/O block size (DOS_RQRIOBLK)

This parameter specifies the size of the communication buffer (requester I/O block size) between DOS/Windows 3.1 applications and their database agents on the database server. The DOS_RQRIOBLK parameter applies only to DOS clients, while the RQRIOBLK parameter applies to non-DOS clients.

Applications use heaps to exchange information between the application and the agent. The heap is also used for row blocking, which retrieves a block of rows in a single operation. This parameter, DOS_RQRIOBLK, is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of application's private address space. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

The memory area is allocated when the application opens a cursor. Blocking will be used depending on the options specified when precompiling and binding the application. The default size is 4 KB. It is independent of the protocol used to connect the client and server.

You increase the value of `DOS_RQRIOBLK` for the following conditions/scenarios:

- A single SQL statement transmits large data (for example, large object data).
- Number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, as this will increase the size of the working set memory for each connection, a trade-off has to be reached.
- Larger record blocks may also cause more fetch requests than are actually required by the application. You can control the fetch requests by using the OPTIMIZE FOR clause on the SELECT statement in your application.

Chapter 7. Tuning application performance

When you are trying to design a new database system or analyze an existing database system, one of the most important considerations is your application design. Even though your database is well designed and tuned, inappropriate design of applications may cause performance problems. If your application has a design problem, fixing this often improves the application performance much more than tuning the configuration parameters of DB2 UDB.

For example, SQL is a high-level language with much flexibility. Different `SELECT` statements can be written to retrieve the same data; however, the performance can vary for the different forms of `SELECT` statements. This is because one statement may have a higher processing cost than another. In such a case, you should choose the SQL statement which has the lower processing cost, so that the application will have good performance.

In this section, we will discuss application design considerations to obtain better performance. These include:

- Tips to write better SQL statements
- Minimizing data transmission between applications and the database
- Considerations for embedded SQL programs
- Considerations for ODBC/CLI programs
- Concurrency on database objects

Study this chapter and apply these considerations when you develop or evaluate your database applications.

7.1 Writing better SQL statements

DB2 UDB provides the SQL compiler which creates the compiled form of SQL statements. When the SQL compiler compiles SQL statements, it rewrites them into a form that can be optimized more easily. This is known as *query rewrite*.

The SQL compiler then generates many alternative execution plans for satisfying the user's request. It estimates the execution cost of each alternative plan using the statistics for tables, indexes, columns, and functions, and chooses the plan with the smallest execution cost. This is known as *query optimization*.

It is important to note that the SQL compiler (including the query rewrite and optimization phases) must choose an access plan that will produce the result set for the query you have coded. Therefore, as noted in many of the following guidelines, you should code your query to obtain only the data that you need. This ensures that the SQL compiler can choose the best access plan for your needs.

Guidelines for using a `SELECT` statement are these:

- Specify only needed columns.
- Limit the number of rows.
- Specify the `FOR UPDATE` clause if applicable.
- Specify the `OPTIMIZED FOR n ROWS` clause.
- Specify the `FETCH FIRST n ROWS ONLY` clause if applicable.
- Specify the `FOR FETCH ONLY` clause if applicable.
- Avoid numeric data type conversion.

Each of these guidelines are further explored in the next section.

7.1.1 Specify only needed columns in the select list

Specify only those columns that are needed in the select list. Although it may be simpler to specify all columns with an asterisk (*), needless processing and returning of unwanted columns can result.

7.1.2 Limit the number of rows by using predicates

Limit the number of rows selected by using predicates to restrict the answer set to only those rows that you require. There are four categories of predicates, each with its own processing cost. The category is determined by how and when that predicate is used in the evaluation process. These categories are listed below, ordered in terms of performance, starting with the most favorable:

- Range delimiting predicates
- Index SARGable predicates
- Data SARGable predicates
- Residual predicates

Note

SARGable refers to something that can be used as a search argument.

Range delimiting predicates are those used to bracket an index scan. They provide start and/or stop key values for the index search. Index SARGable predicates are not used to bracket a search, but can be evaluated from the index because the columns involved in the predicate are part of the index key. For example, assume that an index has been defined on the `NAME`, `DEPT`, and `YEARS` columns of the `STAFF` table, and you are executing the following select statement:

```
SELECT name, job, salary FROM staff
    WHERE name  = 'John'
          dept  = 10
          years > 5
```

The first two predicates (`name='John', dept=10`) would be range delimiting predicates, while `years > 5` would be an index SARGable predicate, as the start key value for the index search cannot be determined by this information only. The start key value may be 6, 10, or even higher. If the predicate for the `years` column is `years => 5`, it would be a range delimiting predicate, as the index search can start from the key value 5.

```
SELECT name, job, salary FROM staff
    WHERE name  = 'John'
          dept  = 10
          years => 5
```

The database manager will make use of the index data in evaluating these predicates, rather than reading the base table. These range delimiting predicates and index SARGable predicates reduce the number of data pages accessed by reducing the set of rows that need to be read from the table. Index SARGable predicates do not affect the number of index pages that are accessed.

Data SARGable predicates are the predicates that cannot be evaluated by the Index Manager, but can be evaluated by Data Management Services (DMS). Typically, these predicates require the access of individual rows from a base table. If required, Data Management Services will retrieve the columns needed to evaluate the predicate, as well as any others to satisfy the columns in the `SELECT` list that could not be obtained from the index.

For example, assume that a single index is defined on the `projno` column of the `project` table but not on the `deptno` column, and you are executing the following query:

```
SELECT projno, projname, repemp FROM project
    WHERE deptno='D11'
    ORDER BY projno
```

The predicate `deptno='D11'` is considered data SARGable, because there are no indexes on the `deptno` column, and the base table must be accessed to evaluate the predicate.

Residual predicates, typically, are those that require I/O beyond the simple accessing of a base table. Examples of residual predicates include those using quantified sub-queries (sub-queries with `ANY`, `ALL`, `SOME`, or `IN`), or reading `LONG VARCHAR` or large object (LOB) data (they are stored separately from the table).

These predicates are evaluated by Relational Data Services (RDS). Residual predicates are the most expensive of the four categories of predicates.

As residual predicates and data SARGable predicates cost more than range delimiting predicates and index SARGable predicates, you should try to limit the number of rows qualified by range delimiting predicates and index SARGable predicates whenever possible.

Let us briefly look at the following DB2 components: Index Manager, Data Management Service, and Relational Data Service. Figure 41 shows each DB2 component and where each category of predicates is processed.

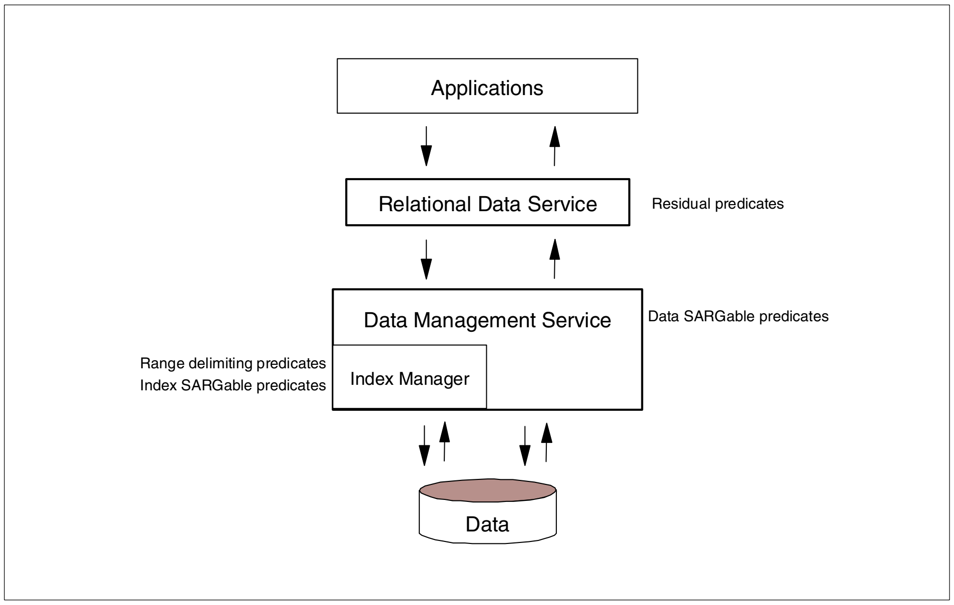


Figure 41. DB2 UDB components and predicates

Note

Figure 41 provides a simplified explanation. Actually, DB2 has more components than are shown in this diagram.

Relational Data Service (RDS) receives SQL requests from applications and returns the result set. It sends all predicates to Data Management Service (DMS) except residual predicates. Residual predicates are evaluated by Relational Data Service (RDS).

DMS evaluates data SARGable predicates. Also, if the select list has columns which cannot be evaluated by the index search, DMS scans data pages directly.

Index Manager receives range delimiting predicates and index SARGable predicates from DMS, evaluates them, and then returns row identifiers (RIDs) to the data page to DMS.

7.1.3 Specify the FOR UPDATE clause

If you intend to update fetched data, you should specify `FOR UPDATE` clause in the `SELECT` statement of the cursor definition. By doing this, the database manager can initially choose appropriate locking levels, for instance, U (update) locks instead of S (shared) locks. Thus you can save the cost to perform lock conversions from S locks to U locks when the succeeding `UPDATE` statement is processed.

The other benefit to specifying the `FOR UPDATE` clause is that can decrease the possibility of deadlock. As we will discuss later in 7.6.8.2, "Deadlock behavior" on page 269, deadlock is a situation in which more than one application is waiting for another application to release a lock on data, and each of the waiting applications is holding data needed by other applications through locking. Let us suppose two applications are trying to fetch the same row and update it simultaneously in the following order:

1. Application1 fetches the row.
2. Application2 fetches the row.
3. Application1 updates the row.
4. Application2 updates the row.

On step 4, Application2 should wait for Application1 to complete the update and release the held lock, and then start its updating. However, if you do not specify the `FOR UPDATE` clause when declaring a cursor, Application1 acquires and holds an S (shared) lock on the row (step 1). That means the second application can also acquire and hold an S lock without lock-waiting (step 2). Then the first application tries to get a U (update) lock on the row to process an `UPDATE` statement, but must wait for the second application to release its holding S lock (step 3). Meanwhile the second application also tries to get a U lock and gets into the lock-waiting status due to the S lock held by the first application (step 4). This situation is a deadlock, and the transaction of the first or second application will be rolled back (see Figure 42).

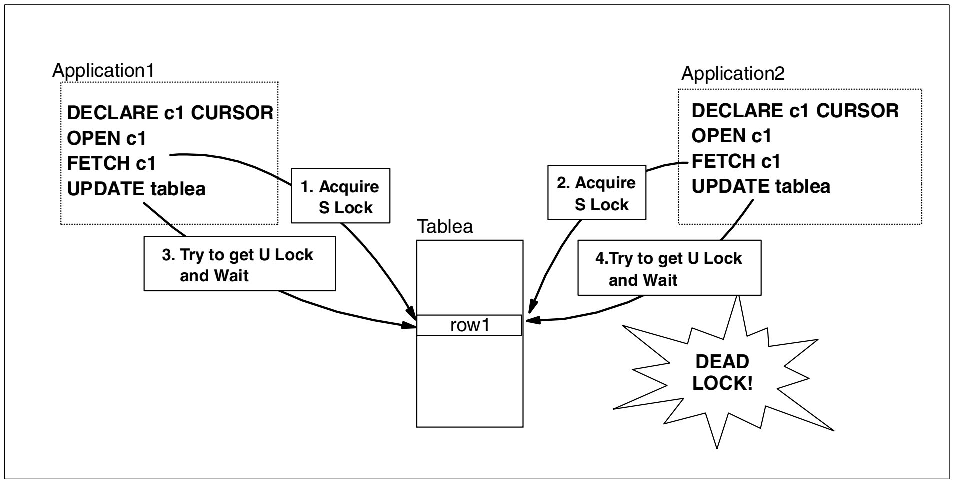


Figure 42. Deadlock between two applications updating same data

If you specify the FOR UPDATE clause in the DECLARE CURSOR statement, the U lock will be imposed when Application1 fetches the row, and Application2 will wait for Application1 to release the U lock. Thus, no deadlock will occur between the two applications.

Note

In this example, we assume either of the two applications does not use the isolation level UR (Uncommitted Read). We will discuss isolation levels in 7.6, "Concurrency" on page 258.

Here is an example of how to use the UPDATE OF clause in a SELECT statement.

```
EXEC SQL DECLARE c1 CURSOR FOR select * from employee
   FOR UPDATE OF job;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO...;
if ( strcmp (change,"YES") == 0)
EXEC SQL UPDATE employee SET job=:newjob
  WHERE CURRENT OF c1;
EXEC SQL CLOSE c1;
```

For CLI programs, you can set `SQL_MODE_READ_WRITE` to the DB2 CLI connection attribute `SQL_ATTR_ACCESS_MODE` using `SQLSetConnectAttr()` the function to achieve the same results. Refer to the `SQLSetConnectAttr()` section of the *Call Level Interface Guide and Reference*, SC09-2950 for more information.

7.1.4 Specify the OPTIMIZE FOR n ROWS clause

Specify the `OPTIMIZE FOR n ROWS` clause in the `SELECT` statement when the number of rows you want to retrieve is significantly less than the total number of rows that could be returned. Use of the `OPTIMIZE FOR` clause influences query optimization based on the assumption that `n` rows will be retrieved. This clause also determines the number of rows that are blocked in the communication buffer.

```
SELECT projno,projname,repemp FROM project
      WHERE deptno='D11' OPTIMIZE FOR 10 ROWS
```

Row blocking is a technique that reduces database manager overhead by retrieving a block of rows in a single operation. These rows are stored in a cache, and each `FETCH` request in the application gets the next row from the cache. If you specify `OPTIMIZE FOR 10 ROWS`, a block of rows is returned to the client every ten rows.

Note

The `OPTIMIZE FOR n ROWS` clause does not limit the number of rows that can be fetched or affect the result in any way other than performance. Using `OPTIMIZE FOR n ROWS` can improve the performance if no more than n rows are retrieved, but may degrade performance if more than n rows are retrieved.

7.1.5 Specify the FETCH FIRST n ROWS ONLY clause

Specify the `FETCH FIRST n ROWS ONLY` clause if you do not want the application to retrieve more than `n` rows, regardless of how many rows there might be in the result set when this clause is not specified. This clause cannot be specified with the `FOR UPDATE` clause.

For example, with the following coding, you will not receive more than 5 rows:

```
SELECT projno,projname,repemp FROM project
    WHERE deptno='D11'
    FETCH FIRST 5 ROWS ONLY
```

The `FETCH FIRST n ROWS ONLY` clause also determines the number of rows that are blocked in the communication buffer. If both the `FETCH FIRST n ROWS ONLY` and `OPTIMIZE FOR n ROWS` clause are specified, the lower of the two values is used to determine the communication buffer size.

7.1.6 Specify the FOR FETCH ONLY clause

If you have no intention of updating rows retrieved by a `SELECT` statement, specify the `FOR FETCH ONLY` clause in the `SELECT` statement. It can improve performance by allowing your query to take advantage of row blocking. It can also improve data concurrency since exclusive locks will never be held on the rows retrieved by a query with this clause specified (see 7.6, “Concurrency” on page 258).

Note

Instead of the `FOR FETCH ONLY` clause, you can also use the `FOR READ ONLY` clause. ‘`FOR READ ONLY`’ is a synonym for ‘`FOR FETCH ONLY`’.

7.1.7 Avoid data type conversions

Data type conversions (particularly numeric data type conversions) should be avoided whenever possible. When two values are compared, it may be more efficient to use items that have the same data type. For example, suppose you are joining `TableA` and `TableB` using column `A1` of `TableA` and column `B1` of `TableB` as in the following example.

```
SELECT * FROM TableA,TableB WHERE A1=B1
```

If columns `A1` and `B1` are the same data type, no data type conversion is required. But if they are not the same data type, a data type conversion occurs to compare values at run time and it might affect the performance. For example, if `A1` is a decimal column and B1 is an integer column and each has a value ‘123’, data type conversion is needed, as `TableA` stores it as x‘123C’, whereas `TableB` stores it as x‘7B’.

Also, inaccuracies due to limited precision may result when data type conversions occur.

7.1.7.1 Other considerations for data types

DB2 UDB allows you to use various data types. You can use SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, and DOUBLE for numeric data; CHAR, VARCHAR, LONG VARCHAR, CLOB for character data; GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, and DBCLOB for the double byte character data, and so on. As the amount of database storage and the cost to process varies depending on the data type, you should choose the appropriate data type.

The following are guidelines when choosing a data type:

- Use character (CHAR) rather than varying-length character (VARCHAR) for short columns. The varying-length character data type can save database storage when the length of data values varies, but there is a cost to check the length of each data value.
- Use VARCHAR or VARGRAPHIC rather than LONG VARCHAR or LONG VARGRAPHIC. The maximum length for VARCHAR and LONG VARCHAR columns, VARGRAPHIC and LONG VARGRAPHIC are almost same (32,672 bytes for VARCHAR, 32,700 bytes for LONG VARCHAR, 16,336 characters for VARGRAPHIC, and 16,350 characters for LONG VARGRAPHIC), while LONG VARCHAR and LONG VARGRAPHIC columns have several restrictions. For example, data stored in LONG VARCHAR or LONG VARGRAPHIC columns is not buffered in the database buffer pool. See 4.1.1.4, "Use VARCHAR or LONG VARCHAR" on page 89 for further description.
- Use integer (SMALLINT, INTEGER, BIGINT) rather than floating-point number (REAL or DOUBLE) or decimal (DECIMAL) if you do not need to have the fraction part. Processing cost for integers is much more inexpensive.
- Use date-time (DATE, TIME, TIMESTAMP) rather than character (CHAR). Date-time data types consume less database storage, and you can use some built-in functions for date-time data types such as YEAR and MONTH.
- Use numeric data types rather than character.

For detailed information about the supported data types, refer the *DB2 UDB SQL Reference*, SC09-2974.

7.2 Minimize data transmission

Network costs are often the performance gating factor for applications. A good first step in investigating this is to run the application, or individual queries from it, locally on the server and see how much faster it runs. That, and the use of network monitoring tools, can indicate if network tuning or a faster network is called for. Note that here “network” includes the local case, since even though local connections have much less overhead than a network protocol, the overhead is still significant.

Note

If there is a relatively small set of queries in the application, db2batch is a good tool to investigate the queries. See 5.4.5, “The db2batch utility” on page 173.

If the network itself is in good shape, you should focus on reducing the number of calls that flow from the application to the database (even for local applications).

There are several ways to reduce network costs. Here we will introduce two ways which involve having multiple actions take place through one call.

7.2.1 Compound SQL

Compound SQL is a technique to build one executable block from several SQL statements. When compound SQL is being executed, each SQL statement in the block is executed individually, but the number of requests transmitting between client and server can be reduced.

Here is an example executing two UPDATE statements and one INSERT statement using compound SQL:

```
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
    UPDATE tablea SET cola = cola * :var1;
    UPDATE tableb SET colb = colb + :var2;
    INSERT INTO tablec (colc,cold,cole) VALUES (:i,:j,0);
  END COMPOUND;
```

When you execute compound SQL, you can choose two types of compound SQL, atomic and non-atomic. The type determines how the entire block is handled when one or more SQL statements in the block happen to end in error. The type of the example above is atomic.

Atomic: If one of the statements in the block ends in error, the entire block is considered to have ended in and error, and any changes made to the database within the block will be rolled back.

Non-atomic: When all statements have completed, the application receives a response. Even if one or more statements in the block end in error, the database manager will attempt to execute all statements in the block. If the unit of work containing the compand SQL is rolled back, then all changes made to the database within the block will be rolled back.

You can also use compound SQL to improve performance of the IMPORT utility, which repeats data inserts. When you specify `MODIFIED BY COMPOUND=x` option (x is the number of inserts compounded into one block), the IMPORT utility builds a block from multiple inserts. You will gain significant performance improvement from this option. See the 8.3, “IMPORT utility” on page 279 for more information.

7.2.2 Stored procedures

A stored procedure resides on a database server, where it executes, and accesses the database locally to return information to client applications. Using stored procedures allows a client application to pass control to a stored procedure on the database server. This allows the stored procedure to perform intermediate processing on the database server, without transmitting unnecessary data across the network. Only those records that are actually required are transmitted. This can result in reduced network traffic and better overall performance.

A stored procedure also saves the overhead of having a remote application pass multiple SQL statements to a database on a server. With a single statement, a client application can call the stored procedure, which then performs the database access work and returns the results to the client application. The more SQL statements that are grouped together for execution in the stored procedure, the larger the savings resulting from avoiding the overhead associated with network flows for each SQL statement when issued from the client.

To create a stored procedure, you must write the application in two separate procedures. The calling procedure is contained in a client application and executes on the client. The stored procedure executes at the location of the database on the database server.

Note

Since the release of Version 7.1, you can write a stored procedure using SQL statements within a `CREATE PROCEDURE` statement.

If your system has a large number of stored procedure requests, you should consider tuning some database manager configuration parameters, which are explored in the following sections.

7.2.2.1 Nested stored procedures

Nested stored procedures are stored procedures that call another stored procedure. Up to 16 levels of nested stored procedure calls are supported. By using this technique, you can implement more complex procedural logic without increasing client-side overhead.

7.2.2.2 Keep stored procedure processes

You can define two types of stored procedure, not-fenced or fenced. Not-fenced procedures run in the database manager operating environment's process, whereas fenced procedures run in other processes to be insulated from the internal resources on the database manager. Therefore, not-fenced procedures can provide better performance, but could cause problems to the database manager if they contain bugs.

In an environment in which the number of fenced stored procedure requests is large, you should consider keeping a stored procedure process (called a DARI process) idle after a stored procedure call is completed. This consumes additional system resources; however, the performance of a stored procedure can be improved because the database manager does not have to create a new DARI process for each stored procedure request.

To keep DARI processes idle, set the database manager configuration parameter `KEEPDARI` to `YES`. This is the default value. You can set this configuration parameter by using the Control Center, or by executing the following command from the Command Line Processor in the DB2 Server:

```
db2 UPDATE DBM CFG USING keepdari yes
```

If you are developing a stored procedure, you may want to modify and test loading the same stored procedure library a number of times. This default setting, `KEEPDARI=YES` may interfere with reloading the library, and therefore you need to stop the database manager to load the modified stored procedure library. It is best to change the value of this keyword to `no` while developing stored procedures, and then change it back to `yes` when you are ready to load the final version of your stored procedure.

Note

For Java stored procedures, even though you set KEEPDARI=YES, you can force DB2 to load new classes instead of stopping the database manager. See “Refresh Java stored procedure classes” on page 243.

You can use the database manager configuration parameter `MAXDARI` to control the maximum number of the DARI processes that can be started at the DB2 server. The default value is dictated by the maximum number of coordinating agents, which is specified by the `MAX_COORDAGENTS` database manager configuration parameter. Since no more than one DARI process can be active per coordinating agent, you cannot set a bigger value than the maximum number of coordinating agents. Make sure this parameter is set to the number of applications allowed to make stored procedure calls at one time.

You can set the `NUM_INITDARIS` database manager configuration parameter to specify the number of initial idle DARI processes when the database manager is started. The default value is 0. By setting this parameter and the `KEEPDARI` parameter, you can reduce the initial startup time for fenced stored procedures.

7.2.2.3 Load Java Virtual Machine for stored procedure processes

When your application calls a fenced Java stored procedure, the DARI process for the stored procedure loads the Java Virtual Machine (JVM). Therefore, if you want to reduce the initial startup time for fenced Java stored procedures, you should set the database manager configuration parameter `INITDARI_JVM=YES` so that each fenced DARI process loads the JVM when starting. This will reduce the initial startup time for fenced Java stored procedures when used in conjunction with the `NUM_INITDARI` parameter and `KEEPDARI` parameter.

This parameter could increase the initial load time for non-Java fenced stored procedures, as they do not require the JVM.

7.2.2.4 Refresh Java stored procedure classes

When a fenced DARI process executes a Java stored procedure, the process loads the JVM, and the JVM locks the Java routine class for the stored procedure. If you set `KEEPDARI=NO`, the lock will be released after the stored procedure is completed and the DARI process is terminated; however, if you set `KEEPDARI=YES`, the DARI process and the JVM is up even after the stored procedure terminates. That means even though you update the Java routine class for the stored procedure, DB2 will continue to use the old version of the class.

To force DB2 to load the new class, you have two options. One is restarting the database manager, which may be not always acceptable. The other option is executing a `SQLJ.REFRESH_CLASSES` statement. By executing this command, you can replace Java stored procedure classes without stopping the database manager even if you set `KEEPDARI=YES`. Execute the following from the command line:

```
db2 CALL SQLJ.REFRESH_CLASSES()
```

Note

You cannot update a not-fenced Java stored procedure without stopping and restarting the database manager.

7.3 Embedded SQL program

Embedded SQL programs are those statements in which SQL statements are embedded. You can write embedded SQL programs in the C/C++, COBOL, FORTRAN, Java (SQLJ), and REXX programming languages, and enable them to perform any task supported by SQL, such as retrieving or storing data.

7.3.1 Static SQL

There are two types of embedded SQL statements: static and dynamic. Static SQL statements are ones where the SQL statement type and the database objects accessed by the statement, such as column names, are known prior to running the application. The only unknowns are the data values the statement is searching for or modifying.

You must pre-compile and bind such applications to the database so that the database manager analyzes all of static SQL statements in a program, determines its access plan to the data, and store the ready-to-execute application package before executing the program. Because all logic to execute SQL statements is determined before executing the program, static SQL programs have the least run-time overhead of all the DB2 programming methods, and execute faster.

To prepare all SQL statements in a static embedded SQL program, all database objects being accessed must exist when binding the package. If you want to bind the package when one or more database objects are missing, specify the option `SQLERROR CONTINUE` in conjunction with `VALIDATE RUN` in the `BIND` or `PREP` command. Though you encounter errors for SQL statements which try to access missing database objects, the package will be bound. For the SQL statements which had errors during the bind, the rebind process will be performed at execution time. If you want to have the least run-time overhead, make sure that all database objects being accessed exist during the bind.

7.3.1.1 When and how the access plan is determined

The DB2 optimizer determines the best access plans for static SQL programs when the bind operation is performed. The determination is done based on the statistical information stored in the system catalog. Obsolete statistical information may lead the DB2 optimizer to select inefficient access plans and may cause performance problems. Therefore, it is very important to collect up-to-date statistical information using the `RUNSTATS` utility before binding packages.

Since static SQL statements are processed based on the access plans determined during the bind, there might be better access plans if you make numerous changes to the database after the bind. For example, assuming you have a very small table with an index and your application has static SQL statements retrieving data from the index keys, then the application tends to access the table using table scans rather than index scans because the size is too small to benefit from index scans; however, if the table considerably grows up, index scans are preferable. In such a case, you should consider executing `RUNSTATS` to refresh the table and index's statistical information stored in the system catalog, and execute the `REBIND` command to take new access plans for the static SQL statements.

There are various forms of `RUNSTATS`, but a good default to use is:

```
RUNSTATS ON TABLE xxx AND DETAILED INDEXES ALL
```

Adding the `WITH DISTRIBUTION` clause can be a very effective way to improve access plans where data is not uniformly distributed.

Access path to volatile tables

If you have volatile tables whose size can vary from empty to quite large at run time, relying on the statistics collected by `RUNSTATS` to generate an access path to a volatile table can be misleading. For example, if the statistics were collected when the volatile table was empty the optimizer tends to favor accessing the volatile table using a table scan rather than an index scan. If you know a table is volatile, you can let the DB2 optimizer to select index scans regardless of the existing statistics of these tables by executing the `ALTER TABLE` statement with the `VOLATILE` option.

```
ALTER TABLE tablename VOLATILE
```

When a table is known to be volatile to the optimizer, it will favor index scans rather than table scans. This means that access paths to a volatile table will not depend on the existing statistics on this table. These statistics will be ignored by the optimizer because they can be misleading, in the sense that they are static and do not reflect the current content of the table.

To deactivate the volatile option and let the DB2 optimizer choose access paths based on the existing statistics, execute the following statement:

```
ALTER TABLE tablename NOT VOLATILE
```

7.3.2 Dynamic SQL

Dynamic SQL statements are ones that your application builds and executes at run time. An interactive application that prompts the end user for key parts of an SQL statement, such as the names of the tables and columns to be searched, is a good example of dynamic SQL. The application builds the SQL statement while it is running, and then submits the statement for processing. Generally, dynamic SQL statements are well-suited for applications that run against a rapidly changing database where transactions need to be specified at run time.

7.3.2.1 When and how the access plan is determined

Dynamic embedded SQL requires the precompile, compile, and link phases of application development. However, the binding or selection of the most effective data access plan is performed at program execution time, as the SQL statements are *dynamically prepared*.

An embedded dynamic SQL programming module will have its data access method determined during the statement preparation and will utilize the database statistics available at query execution time. Choosing an access plan at program execution time has some advantages as well as a drawback.

Some of the advantages are:

- Current database statistics are used for each SQL statement.
- Database objects do not have to exist before run time.
- This method is more flexible than using static SQL statements.

One drawback is that dynamic SQL statements can take more time to execute, since queries are optimized at run time. To improve your dynamic SQL program's performance, the following are key:

- Execute `RUNSTATS` after making significant updates to tables or creating indexes.
- Minimize preparation time for dynamic SQL statements.

Keeping statistics information up-to-date helps the DB2 optimizer to choose the best access plan. You do not need to rebind packages for dynamic SQL programs after executing `RUNSTATS` since access plan is determined at run time.

In the following section, we will discuss how to minimize preparation time for dynamic SQL statements.

7.3.2.2 Avoid repeated PREPARE statements

When an SQL statement is prepared, it is parsed, optimized, and made ready to be executed. The cost of preparing can be very significant, especially relative to the cost of executing very simple queries (it can take longer to prepare such queries than to execute them). To improve the performance of dynamic SQL, the global package cache was introduced in DB2 UDB Version 5.0. The generated access plan for an SQL statement is stored in the global package cache, and it can be reused by the same or other applications. Thus, if the same exact statement is prepared again, the cost will be minimal. However, if there is any difference in syntax between the old and new statements, the cached plan cannot be used.

For example, suppose the application issues a `PREPARE` statement for the statement "`SELECT * FROM EMPLOYEE WHERE empno = '000100'` ", then issues another `PREPARE` statement for "`SELECT * FROM EMPLOYEE WHERE empno = '000200'` " (the same statement but with a different literal value). The cached plan for the first statement cannot be reused for the second, and the latter's `PREPARE` time will be non-trivial. See the following example:

```
strcpy (st1,"SELECT * FROM EMPLOYEE WHERE empno='000100'");
strcpy (st2,"SELECT * FROM EMPLOYEE WHERE empno='000200'");
EXEC SQL PREPARE s1 FROM :st1;
EXEC SQL PREPARE s2 FROM :st2;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL DECLARE c2 CURSOR FOR s2;
EXEC SQL OPEN c1;
EXEC SQL OPEN c2;
...
```

The solution is to replace the literal '`000100`' by a question mark (?), issue a `PREPARE`, declare the cursor for the statement, assign the literal when opening the cursor. By changing the program variable(s) appropriately before each OPEN statement, you can reuse the prepared statement. See the following example:

```
strcpy (st,"SELECT * FROM EMPLOYEE WHERE empno='?'");
EXEC SQL PREPARE s1 FROM :st;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL DECLARE c2 CURSOR FOR s1;
strcpy (parmvar1,"000100");
strcpy (parmvar2,"000100");
EXEC SQL OPEN c1 using :parmvar1;
...
EXEC SQL OPEN c2 using :parmvar2;
...
```

Parameter markers can and should be used, not just for `SELECT`, but also for repeated executions of `INSERT`, `UPDATE`, or `DELETE` statements. For example, if your application is using `EXECUTE IMMEDIATE` to execute multiple statements that differ only in the literal values they contain, those `EXECUTE IMMEDIATE` statements should be replaced by `PREPARE` and `EXECUTE` statements using parameter markers. See the following example to read records from a file and insert them into a table:

```
for ( end of file ) {
   ...
   //Read a record from the input file;
   //Generate INSERT statement to store the record
   //into a table and save the statement into
   //the host variable stmt;
   ...
   EXEC SQL EXECUTE IMMEDIATE :stmt
}
```

In this example, generated INSERT statements are prepared and executed for each record being inserted. If the input file has many rows, preparing all INSERT statements will be expensive. You should change this example as follows:

```
strcpy( stmt,"INSERT INTO tablea VALUES (?,?,?)");
EXEC SQL PREPARE st FROM :stmt;
for ( end of file ) {
   ...
   //Read a record from the input file;
   //Assign read values into the host
   //variables (var1,var2,var3) for the parameter markers;

   EXEC SQL EXECUTE st USING :var1,:var2,:var3;
}
```

This example can complete the insert job faster, since only one INSERT statement is prepared and reused for all rows being inserted.

7.3.2.3 Tune optimization level

Sometimes another cause of long preparation times is the use of a query optimization class that is higher than necessary. That is, the DB2 Optimizer can spend more time finding the best access plan for a query than is justified by a reduction in execution time.

For example if the database has large number of concurrent activity, numerous simple SQL statements to process, and a requirement to perform them in seconds, set the optimization class to a lower value such as 1 or 2 by the SET CURRENT QUERY OPTIMIZATION statement. If you do not set any optimization level in the CURRENT QUERY OPTIMIZATION special register, the DB2 optimizer will table the value set in the DFT_QUERYOPT database configuration parameter.

7.4 Call Level Interface and ODBC

The DB2 Call Level Interface (CLI) is a programming interface that your C and C++ applications can use to access DB2 databases. DB2 CLI is based on the Microsoft Open Database Connectivity Standard (ODBC) specification, and the X/Open and ISO Call Level Interface standards. Many ODBC applications can be used with DB2 without any modifications. Likewise, a CLI application is easily ported to other database servers.

DB2 CLI and ODBC provide a dynamic SQL application development environment. The SQL statements are issued through direct API calls. The DB2 optimizer prepares the SQL statements when the application runs. Therefore, the same advantages as dynamic embedded SQL programs are also true for DB2 CLI and ODBC programs. As we saw in the previous section, the advantages are:

- Current database statistics are used for each SQL statement.
- Database objects do not have to exist before run time.
- More flexible than static SQL statements.

Moreover, DB2 CLI and ODBC applications have the following advantages:

- Can store and retrieve sets of data.
- Can use scrollable and updatable cursors.
- Easy porting to other database platforms.

A drawback to using DB2 CLI and ODBC is that the dynamic preparation of SQL statements can result in slower query execution.

7.4.1 Improve performance of CLI/ODBC applications

Since the DB2 optimizer prepares the SQL statements in CLI/ODBC programs at run time like dynamic SQL programs, the following are considerations to improve performance:

- Execute `RUNSTATS` after making significant updates to tables or creating indexes
- Minimize preparation time for SQL statements

As we have already discussed, since the DB2 optimizer tries to find the best access plan for each SQL statement based on the current statistics information saved in the system catalog, refreshing statistics information using `RUNSTATS` will help the DB2 optimizer to determine the best access plan.

To minimize preparation time for SQL statements in CLI/ODBC programs, you should consider avoiding repeated `PREPARE` statement, and use the appropriate optimization level as discussed in the dynamic embedded SQL program section (see 7.3.2, “Dynamic SQL” on page 245). In the following sections, we will discuss how to avoid repeated `PREPARE` statement, and set optimization level in CLI/ODBC programs. We will also introduce two methods to minimize preparation time for CLI/ODBC applications.

7.4.1.1 Avoid repeated PREPARE statements

When you need to execute multiple statements that differ only in the literal values they contain, you can use `SQLExecDirect` repeatedly for each statements; however, this approach is expensive since each statement is prepared one by one. To avoid preparing similar SQL statements repeatedly, you can use an `SQLPrepare` call instead of multiple `SQLExecDirect`. Your program should perform the following steps:

1. Call an `SQLPrepare` to prepare the SQL statement with parameter markers.
2. Issue an `SQLBindParameter` to bind a program variable to each parameter marker.
3. Issue an `SQLExecute` call to process the first SQL statement.
4. Repeat `SQLBindParameter` and `SQLExecute` as many times as required.

The ready-to-execute package prepared by `SQLPrepare` will be reused for each SQL statement.

7.4.1.2 Tune optimization level

As discussed in the dynamic embedded SQL section, if the database is in an environment such as Online Transaction Processing (OLTP), which typically has numerous simple SQL statements to process, set the optimization class to a lower value such as 1 or 2. To set the optimization level within the application, use `SQLExecDirect` to issue a `SET CURRENT QUERY OPTIMIZATION` statement. To set the same optimization level for all the CLI/ODBC applications on a client, use the `UPDATE CLI CFG` command from the client as in the following example:

```
UPDATE CLI CFG FOR SECTION database1 USING DB2OPTIMIZATION 2
```

This command sets the CLI/ODBC keyword `DB2OPTIMIZATION=2` in the `db2cli.ini` file so that the DB2 optimizer will use the optimization level 2 to optimize SQL statements of all the CLI/ODBC applications accessing the database `database1` from this client.

7.4.1.3 Use an optimized copy of catalog

Many applications written using ODBC or DB2 CLI make heavy use of the system catalog. Since the tables that make up the DB2 catalog contain many columns that are not required by the ODBC driver, ODBC/CLI applications can cause DB2 to retrieve a lot of extraneous data when reading DB2 catalog data pages. Also, the ODBC driver often has to join results from multiple DB2 catalog tables to produce the output required by the ODBC driver’s callable interfaces.

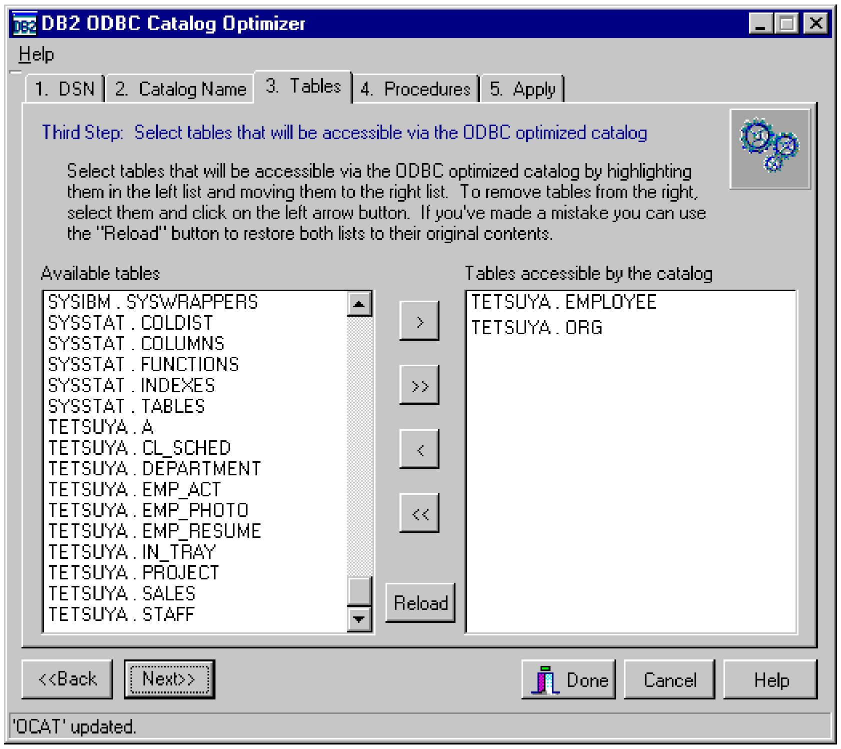
While this does not usually present a problem for databases with a small number of database objects (tables, views, synonyms and so on), it can lead to performance problems when using these applications with larger DB2 databases.

This performance degradation can be attributed to 2 main factors: the amount of information that has to be returned to the calling application and the length of time that locks are held on the catalog tables.

The db2ocat tool solves both problems by creating separate system catalog tables called the *ODBC optimized catalog tables* that has only the columns necessary to support ODBC/CLI operations.

The db2ocat tool is a 32-bit Windows program that can be used on Windows workstations running the DB2 Version 6.1 (or later) client. You can create ODBC optimized catalog tables in DB2 databases on any platform from this tool running on Windows workstations.

Using the db2ocat tool, you can identify a subset of tables and stored procedures that are needed for a particular application and create a ODBC optimize catalog that is used to access data about those tables. Figure 43 shows the db2ocat tool GUI which is used to select tables that will be accessible through the ODBC optimized catalog:



Third Step: Select tables that will be accessible via the ODBC optimized catalog

Select tables that will be accessible via the ODBC optimized catalog by highlighting them in the left list and moving them to the right list. To remove tables from the right, select them and click on the left arrow button. If you've made a mistake you can use the "Reload" button to restore both lists to their original contents.

Available tables

```
SYSIBM . SYSWRAPPERS
SYSSTAT . COLDIST
SYSSTAT . COLUMNS
SYSSTAT . FUNCTIONS
SYSSTAT . INDEXES
SYSSTAT . TABLES
TETSUYA . A
TETSUYA . CL_SCHED
TETSUYA . DEPARTMENT
TETSUYA . EMP_ACT
TETSUYA . EMP_PHOTO
TETSUYA . EMP_RESUME
TETSUYA . IN_TRAY
TETSUYA . PROJECT
TETSUYA . SALES
TETSUYA . STAFF
```

Tables accessible by the catalog

```
TETSUYA . EMPLOYEE
TETSUYA . ORG
```

'OCAT' updated.

Figure 43. The db2ocat tool

An ODBC optimized catalog consists of ten tables with the specified schema name. If you specify OCAT as the schema name during the creation of the ODBC optimized catalog, the following tables will be created:

- OCAT.TABLES
- OCAT.COLUMNS
- OCAT.SPECIALCOLUMNS
- OCAT.TSTATISTICS
- OCAT.PRIMARYKEYS
- OCAT.FOREIGNKEYS
- OCAT.TABLEPRIVILEGES
- OCAT.COLUMNTABLES
- OCAT.PROCEDURES
- OCAT.PROCEDURESCOLUMNS

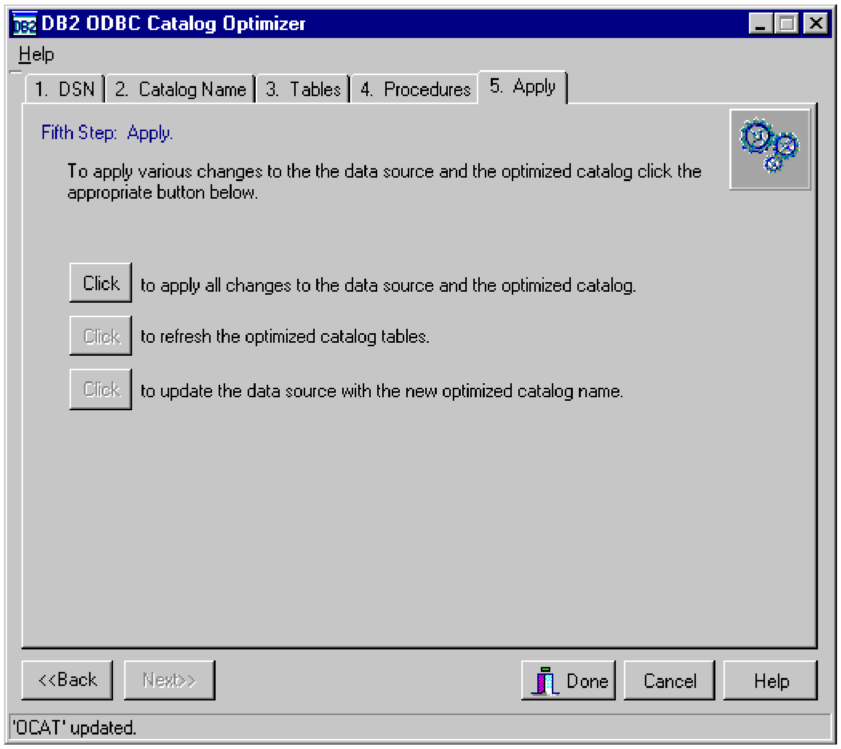
These tables contain only the rows representing database objects you selected and the columns required for ODBC/CLI operations. Moreover, the tables are pre-formatted for the maximum ODBC performance. By using the

ODBC optimized catalog, the ODBC driver does not need to acquire locks on the real system catalog tables or perform join operations for results from multiple tables. Therefore, catalog query times and amount of data returned as a result of these queries are substantially reduced.

You can have multiple ODBC optimized catalogs for different clients. The ODBC optimized catalog is pointed to by the `CLISCHEMA` keyword. If the schema name of the ODBC optimized catalog is `OCAT`, then set `CLISCHEMA=OCAT` in `db2cli.ini` file on the client. You can directly edit the db2cli.ini file or execute the following command:

```
UPDATE CLI CFG FOR SECTION database1 USING CLISCHEMA OCAT
```

The contents in the ODBC optimized catalog is not replicated automatically from the system catalog. You must refresh the ODBC optimized catalog using the db2ocat tool when you perform something which changes the system catalog such as executing `RUNSTATS` or adding new columns (Figure 44).



Fifth Step: Apply.

To apply various changes to the the data source and the optimized catalog click the appropriate button below.

Click to apply all changes to the data source and the optimized catalog.

Click to refresh the optimized catalog tables.

Click to update the data source with the new optimized catalog name.

'OCAT' updated.

Figure 44. The db2ocat tool (refresh ODBC optimized catalog

The db2ocat tool is available at the following site:

`ftp://ftp.software.ibm.com/ps/products/db2/tools/ in the file db2ocat.exe.`

The readme file is available in the `db2ocat.zip` file at the same site.

7.4.1.4 Convert ODBC/CLI into static SQL

As ODBC/CLI applications are dynamic SQL applications, the most effective data access plan of each query is generated at program execution time. This process is expensive since the system catalog tables must be accessed for the resolution for the SQL statements and the statements are optimized. By using the db2ocat tool (see 7.4.1.3, “Use an optimized copy of catalog” on page 250), the cost to access the system catalog can be reduced; however, ODBC/CLI applications and dynamic SQL applications can be still slower than static SQL applications whose SQL statements are ready-to-execute.

In this section, we introduce the method to convert ODBC/CLI applications into static SQL applications. The information of an executed ODBC/CLI application can be captured, and the executable form of statements are stored in the database as a package. Other ODBC/CLI applications can use it like static SQL applications without the preparation cost of the SQL statements.

Note

You need to use DB2 UDB Version 7.1 or later to convert ODBC/CLI applications to static SQL applications.

ODBC/CLI applications run in the following three different modes:

- **Normal mode**

  This is the default value and the traditional ODBC/CLI usage.

- **Capture mode**

  This is the mode used by the database administrator who will run an ODBC/CLI application to capture its connection and statement attributes, SQL statements, and input as well as output SQLDAs. When a connection is terminated, the captured information is saved into an ASCII text file specified by `STATICCAPFILE` keyword in the `db2cli.ini` file. This file should be distributed to other clients as well as the application, and also the package should be created using the `db2cap bind` command, just as you would create a package using the `bind` command for a static SQL application.

- **Match mode**

  This is the mode used by the end user to run ODBC/CLI applications that were pre-bound and distributed by the database administrator. When a connection is established, the captured information associated with the data source name will be retrieved from the local capture file specified by STATICCAPFILE keyword in the db2cli.ini file. If a matching SQL statement is found in the capture file, the corresponding static SQL statement will be executed. Otherwise, the SQL statement will still be executed as a dynamic SQL statement.

These modes are specified using the STATICMODE keyword of the db2cli.ini file as in the following example:

```
 [SAMPLE]
STATICCAPFILE=/home/db2inst1/pkg1.txt
STATICPACKAGE=DB2INST1.ODBCPKG
STATICMODE=CAPTURE
DBALIAS=SAMPLE
```

This example specifies the capture mode. Captured information of the ODBC/CLI application accessed SAMPLE database is saved into the /home/db2inst1/pkg1.txt file. The STATICPACKAGE keyword is used to specify the package name to be later bound by the db2cap bind command.

You can directly edit the db2cli.ini file as shown above, or use the UPDATE CLI CFG command as the following example (the captured text file is shown in Figure 45).

```
UPDATE CLI CFG FOR SECTION sample
     USING STATICCAPFILE /home/db2inst1/pkg1.txt
           STATICMODE CAPTURE
           STATICPACKAGE DB2INST1.ODBCPKG
```

```
[COMMON]
CREATOR=
CLIVERSION=07.01.0000
CONTOKENUR=
CONTOKENCS=
CONTOKENRS=
CONTOKENRR=
CONTOKENNC=

[BINDOPTIONS]
COLLECTION=DB2INST1
PACKAGE=ODBCPKG
DEGREE=
FUNCPATH=
GENERIC=
OWNER=DB2INST1
QUALIFIER=DB2INST1
QUERYOPT=
TEXT=

[STATEMENT1]
SECTNO=
ISOLATION=CS
STMTTEXT=select DEPTNO,DEPTNAME,MGRNO,ADMRDEPT,LOCATION from DEPARTMENT
STMTTYPE=SELECT_CURSOR_WITHHOLD
CURSOR=SQLCURCAPCS1
OUTVAR1=CHAR,3,,FALSE,FALSE,DEPTNO
OUTVAR2=VARCHAR,29,,FALSE,FALSE,DEPTNAME
OUTVAR3=CHAR,6,,FALSE,TRUE,MGRNO
OUTVAR4=CHAR,3,,FALSE,FALSE,ADMRDEPT
OUTVAR5=CHAR,16,,FALSE,TRUE,LOCATION
```

Figure 45. Captured file

Note

Although this captured file has only one SQL statement, you can have more statements in a captured file.

If necessary, you can edit the captured file to change the bind options such as `QUALIFIER`, `OWNER`, and `FUNCPATH`.

Then the `db2cap bind` command should be executed to create a package. The captured file and the database name must be specified as the following example:

```
db2cap bind /home/db2inst1 -d sample
```

The created package name have the suffix number depending on its isolation level. The suffix for the package is one of the following:

- 0 = Uncommitted Read

- 1 = Cursor Stability
- 2 = Read Stability
- 3 = Repeatable Read

In our example, only one package `DB2INST1.ODBCPKG1` will be created, since our example shown in Figure 45 has only one SQL statement using the isolation level Cursor Stability. If the captured file has more than one statement and their isolation levels are different, multiple packages will be created with different suffixes.

You can have more than one captured file to create multiple packages in the same database by executing the `db2cap bind` command for each captured file. Be sure that the `PACKAGE` keyword of each captured file has a different value, since it is used to determine the package name.

Lastly, you should distribute both the captured file and the application to all client machines on which you intend to utilize the pre-bound package. On each client, the `STATICMODE` keyword of the `db2cli.ini` file should be set as `MATCH`, and the captured file should be specified using the `STATICCAPFILE` keyword.

```
[SAMPLE]
STATICCAPFILE=/home/db2inst1/pkg1.txt
STATICMODE=MATCH
DBALIAS=SAMPLE
```

7.5 Java interfaces (JDBC and SQLJ)

DB2 provides support for many different types of Java programs, including applets, applications, servlets, and advanced DB2 server-side features. Java programs that access and manipulate DB2 databases can use the Java Database Connectivity (JDBC) API, and the Embedded SQL for Java (SQLJ) standard. Both of these are vendor-neutral SQL interfaces that provide data access to your application through standardized Java methods.

The greatest benefit of using Java, regardless of the database interface, is its *write once, run anywhere* capability, allowing the same Java program to be distributed and executed on various operating platforms in a heterogeneous environment. And since the two Java database interfaces supported by DB2 are industry open standards, you have the added benefit of using your Java program against a variety of database vendors.

For JDBC programs, your Java code passes *dynamic* SQL to a JDBC driver that comes with DB2. Then, DB2 executes the SQL statements through JDBC APIs which use DB2 CLI, and the results are passed back to your Java code. JDBC is similar to DB2 CLI in that you do not have to precompile or bind a JDBC program, since JDBC uses dynamic SQL.

JDBC relies on DB2 CLI; thus the performance considerations which we discussed in the previous section are also applicable to JDBC applications. See 7.4.1, “Improve performance of CLI/ODBC applications” on page 249.

With DB2 SQLJ support, you can build and run SQLJ programs that contain *static* embedded SQL statements. Since your SQLJ program contains static SQL, you need to perform steps similar to precompiling and binding. Before you compile an SQLJ source file, you must translate it with the SQLJ translator to create native Java source code. After translation, you can create the DB2 packages using the DB2 for Java profile customizer (`db2profc`). Mechanisms contained within SQLJ rely on JDBC for many tasks, like establishing connections.

Since SQLJ contains static SQL statements and their access plans are determined before being executed, the same considerations as static embedded SQL programs are applicable to SQLJ applications. See 7.3.1, “Static SQL” on page 243.

7.6 Concurrency

When many users access the same database, you must establish some rules for the reading, inserting, deleting, and updating of data records. The rules for data access are set by each application connected to a DB2 database and are established using locking mechanisms. Those rules are crucial to guarantee the integrity of the data, but they may decrease concurrency on database objects. On a system with much unnecessary locking, your application may take a very long time to process queries due to lock-waiting, even if the system is rich in hardware resources and well tuned. In this section we will discuss how you can control concurrency appropriately and minimize lock-waits to improve your application’s performance.

To minimize lock-waits, what you should consider first is eliminating unnecessary locks, by doing the following:

- Issue `COMMIT` statements at the right frequency.
- Specify the `FOR FETCH ONLY` clause in the `SELECT` statement.
- Perform `INSERT`, `UPDATE`, and `DELETE` at the end of a unit of work if possible.

- Choose the appropriate isolation level.
- Eliminate next key locks by setting DB2_RR_TO_RS=YES if acceptable.
- Release read locks using the WITH RELEASE option of the CLOSE CURSOR statement if acceptable.
- Avoid lock escalations impacting concurrency by tuning the LOCKLIST and MAXLOCKS database configuration parameters.

Each of these guidelines is further explored in the following sections.

7.6.1 Issue COMMIT statements

Executing COMMIT statements takes overhead due to disk I/O such as flushing logged data into disks; however, since all locks held in a unit of work are released at the end of the unit of work, putting COMMIT statements frequently in your application program improves concurrency. When your application is logically at a point of consistency; that is, when the data you have changed is consistent, put in a COMMIT statement.

Be aware that you should commit a transaction even though the application only reads rows. This is because shared locks are acquired in read-only applications (except for the uncommitted read isolation level, which will be discussed in the next section) and held until the application issues a COMMIT or closes the cursor using the WITH RELEASE option (it will be discussed later in this chapter).

Note

If you opened cursors declared using the WITH HOLD option, locks protecting the current cursor position of them will not be released when a COMMIT is performed. See the pages describing DECLARE CURSOR in the *DB2 UDB SQL Reference*, SC09-2974 for the WITH HOLD option in detail.

7.6.2 Specify FOR FETCH ONLY clause

A query with FOR FETCH ONLY clause never holds exclusive locks on the rows, thus you can improve concurrency using this clause. See 7.1.6, “Specify the FOR FETCH ONLY clause” on page 237.

7.6.3 INSERT, UPDATE, and DELETE at end of UOW

When an application issues an INSERT, UPDATE, or DELETE statement, the application acquires exclusive locks on the affected rows and will keep them until the end of the unit of work. Therefore, perform INSERT, UPDATE, and DELETE at the end of a unit of work if possible. This provides the maximum concurrency.

7.6.4 Isolation levels

DB2 Universal Database provides different levels of protection to isolate the data from each of the database applications while it is being accessed.

These levels of protection are known as *isolation levels*, or locking strategies. Choosing an appropriate isolation level ensures data integrity and also avoids unnecessary locking. The isolation levels supported by DB2 are listed below, ordered in terms of concurrency, starting with the maximum:

- Uncommitted Read
- Cursor Stability
- Read Stability
- Repeatable Read

7.6.4.1 Uncommitted Read

The Uncommitted Read (UR) isolation level, also known as dirty read, is the lowest level of isolation supported by DB2. It can be used to access uncommitted data changes of other applications. For example, an application using the Uncommitted Read isolation level will return all of the matching rows for the query, even if that data is in the process of being modified and may not be committed to the database. You need to be aware that two identical queries may get different results even if they are issued within a unit of work, since other concurrent applications can change or modify those rows that the first query retrieves.

Uncommitted Read transactions will hold very few locks. Thus they are not likely to wait for other transaction to release locks. If you are accessing read-only tables or it is acceptable for the application to retrieve uncommitted data updated by another application, use this isolation level, because it is most preferable in terms of performance.

Note

Dynamic SQL applications using this isolation level will acquire locks on the system catalog tables.

7.6.4.2 Cursor Stability

The Cursor Stability (CS) isolation level locks any row on which the cursor is positioned during a unit of work. The lock on the row is held until the next row is fetched or the unit of work is terminated. If a row has been updated, the lock is held until the unit of work is terminated. A unit of work is terminated when either a COMMIT or ROLLBACK statement is executed.

An application using Cursor Stability cannot read uncommitted data. In addition, the application locks the row that has been currently fetched, and no other application can modify the contents of the current row. As the application locks only the row on which the cursor is positioned, two identical queries may still get different results even if they are issued within a unit of work.

When you want the maximum concurrency while seeing only committed data from concurrent applications, this isolation level should be chosen.

7.6.4.3 Read Stability

The Read Stability (RS) isolation level locks those rows that are part of a result set. If you have a table containing 10,000 rows and the query returns 10 rows, then only 10 rows are locked until the end of the unit of work.

An application using Read Stability cannot read uncommitted data. Instead of locking a single row, it locks all rows that are part of the result set. No other application can change or modify these rows. This means that if you issue a query twice within a unit of work, the second run can retrieve the same answer set as the first. However, you may get additional rows, as another concurrent application can insert rows that match to the query.

Note

Remember that selected rows are locked until the end of the unit of work. Therefore, do not forget to issue a COMMIT (or ROLLBACK) statement even if your application is read-only. A COMMIT (or ROLLBACK) statement will terminate the unit of work and release held locks.

7.6.4.4 Repeatable Read

The Repeatable Read (RR) isolation level is the highest isolation level available in DB2. It locks all rows that an application references within a unit of work, no matter how large the result set. In some cases, the optimizer decides during plan generation that it may get a table level lock instead of locking individual rows, since an application using Repeatable Read may acquire and hold a considerable number of locks. The values of the LOCKLIST and MAXLOCKS database configuration parameters (see 7.6.7.1, “Configure LOCKLIST and MAXLOCKS parameter” on page 266) will affect this decision.

An application using Repeatable Read cannot read uncommitted data of a concurrent application. As the name implies, this isolation level ensures the repeatable read to applications, meaning that a repeated query will get the same record set as long as it is executed in the same unit of work. Since an application using this isolation level holds more locks on rows of a table, or even locks the entire table, the application may decrease concurrency. You should use this isolation level only when changes to your result set within a unit of work are unacceptable.

7.6.4.5 Choosing an isolation level

When you choose the isolation level for your application, decide which concurrency problems are unacceptable for your application and then choose the isolation level which prevents these problems. Remember that the more protection you have, the less concurrency is available.

- Use the Uncommitted Read isolation level only if you use queries on read-only tables, or if you are using only SELECT statements and getting uncommitted data from concurrent applications is acceptable. This isolation level provides the maximum concurrency.
- Use the Cursor Stability isolation level when you want the maximum concurrency while seeing only committed data from concurrent applications.
- Use the Read Stability isolation level when your application operates in a concurrent environment. This means that qualified rows have to remain stable for the duration of the unit of work.
- Use the Repeatable Read isolation level if changes to your result set are unacceptable. This isolation level provides minimum concurrency.

7.6.4.6 Setting an isolation level

The isolation level is defined for embedded SQL statements during the binding of a package to a database using the ISOLATION option of the PREP or the BIND command. The following PREP and BIND examples specify the isolation level as the Uncommitted Read (UR).

```
PREP program1.sqc ISOLATION UR
BIND program1.bnd ISOLATION UR
```

If no isolation level is specified, the default level of Cursor Stability is used.

If you are using the command line processor, you may change the isolation level of the current session using the CHANGE ISOLATION command.

```
CHANGE ISOLATION TO rr
```

For DB2 Call Level Interface (DB2 CLI), you can use the `SQLSetConnectAttr` function with the `SQL_ATTR_TXN_ISOLATION` attribute at run time. This will set the transaction isolation level for the current connection referenced by the `ConnectionHandle`. The accepted values are:

- SQL_TXN_READ_UNCOMMITTED : Uncommitted Read
- SQL_TXN_READ_COMMITTED : Cursor Stability
- SQL_TXN_REPEATABLE_READ : Read Stability
- SQL_TXN_SERIALIZABLE : Repeatable Read

You can also set the isolation level using the `TXNISOLATION` keyword of the DB2 CLI configuration as follows:

```
UPDATE CLI CFG FOR SECTION sample USING TXNISOLATION 1
```

The following values can be specified for the `TXNISOLATION` keyword: 1, 2, 4, 8, or 32. Here are their meanings:

- 1 = Uncommitted Read
- 2 = Cursor Stability (default)
- 4 = Read Stability
- 8 = Repeatable Read

You can use the DB2 CLI configuration for JDBC applications as well. If you want to specify the isolation level within the JDBC application program, use the `setTransactionIsolation` method of `java.sql.Connection`. The accepted values are:

- TRANSACTION_READ_UNCOMMITTED : Uncommitted Read
- TRANSACTION_READ_COMMITTED : Cursor Stability
- TRANSACTION_REPEATABLE_READ : Read Stability
- TRANSACTION_SERIALIZABLE : Repeatable Read

7.6.5 Eliminate next key locks

Next key locking is a mechanism to support the Repeatable Read isolation level. If an application modifies a table using such operations as INSERT, DELETE, or UPDATE, the database manager will obtain key locks on the next higher key value than the modified key so that other applications using Repeatable Read can get the same result sets as long as they executes queries in the same unit of work.

However, if you do not have any applications using Repeatable Read, there is no point in using the next key locking mechanism, and next key locks may cause lock-contention. In this case, you should set `DB2_RR_TO_RS=YES` and eliminate next locking as follows:

```
db2set DB2_RR_TO_RS=YES
```

You may significantly improve concurrency on your database objects by setting this registry variable.

This setting affects the instance level, and you need to stop and start the database manager after changing the value.

7.6.6 Close cursor with release

You should use the Read Stability isolation level when qualified rows have to remain stable for the duration of the unit of work. If changes to your result set are unacceptable, you should use the Repeatable Read isolation level. When using Read Stability or Repeatable Read, more locks are hold than using Cursor Stability or Uncommitted Read.

If you look at your application using Read Stability or Repeatable Read, you may find that all queries in the application do not need the protection which Read Stability or Repeatable Read provides, that means, there may be queries which can release locks before the end of the unit of work. For such queries, use a `CLOSE CURSOR` statement that includes the `WITH RELEASE` clause when closing the cursor.

```
CLOSE c1 WITH RELEASE
```

If the `WITH RELEASE` clause is specified, all read locks (if any) that have been held for the cursor will be released. If it is not specified, held locks will not be released until the unit of work ends.

The `WITH RELEASE` clause has no effect for cursors that are operating under the CS or UR isolation levels. When specified for cursors operating under RS or RR isolation levels, the `WITH RELEASE` clause ends some of the guarantees of those isolation levels, because all read locks will be released before the end of the unit of work. An RS and an RR cursor may experience the nonrepeatable read phenomenon, which means that if you open a cursor, fetch rows, close the cursor with `WITH RELEASE` clause, reopen the cursor, and fetch rows again, then the second query can retrieve the different answer set as the first because other applications can update the rows that match to the query. An RR cursor may experience the phantom read phenomenon as well. After closing the cursor with the `WITH RELEASE` clause, the second query can retrieve the additional rows which were not returned by the first query, because other applications can insert rows that match to the query.

If a cursor that is originally RR or RS is reopened after being closed using the `WITH RELEASE` clause, then new read locks will be acquired.

7.6.7 Lock escalation

If your application acquires and holds locks on almost of all rows in one table, it may be better to have one lock on the entire table. Each database allocates a memory area called a *lock list*, which contains all locks held by all applications concurrently connected to the database.

Each lock requires 72 bytes of memory for an object that has no other locks held on it, or 36 bytes of memory for an object that has existing locks held on it. If a number of row locks can be replaced with a single table lock, the locking storage area can be used by other applications.

When DB2 converts the row locks to a table lock on your behalf, this is called *lock escalation*. DB2 will perform lock escalation to avoid resource problems by too many resources being held for the individual locks.

Two database configuration parameters have a direct effect on lock escalation. They are:

- LOCKLIST — defines the amount of memory allocated for the locks.
- MAXLOCKS — defines the percentage of the total lock list permitted to be allocated to a single application.

There are two different situations for lock escalation:

- One application exceeds the percentage of the lock list as defined by the `MAXLOCKS` configuration parameter. The database manager will attempt to free memory by obtaining a table lock and releasing row locks for this application.
- Many applications connected to the database fill the lock list by acquiring a large number of locks. DB2 will attempt to free memory by obtaining a table lock and releasing row locks.

Also note that the isolation level used by the application has an effect on lock escalation:

- Cursor Stability will acquire row level locks initially. If required, table level locks can be obtained in such a case as updating many rows in a table. Usually, a very small number of locks are acquired by each cursor stability application, since they only have to guarantee the integrity of the data in the current row.

- Read Stability locks all rows in the original result set.
- Repeatable Read may or may not obtain row locks on all rows read to determine the result set. If it does not, then a table lock will be obtained instead.

If a lock escalation is performed, from row to table, the escalation process itself does not take much time; however, locking entire tables decreases concurrency, and overall database performance may decrease for subsequent accesses against the affected tables.

Once the lock list is full, performance can degrade, since lock escalation will generate more table locks and fewer row locks, thus reducing concurrency on shared objects in the database. Your application will receive an SQLCODE of -912 when the maximum number of lock requests has been reached for the database.

7.6.7.1 Configure LOCKLIST and MAXLOCKS parameter

To avoid decreasing concurrency due to lock escalations or errors due to a lock list full condition, you should set appropriate values for both the `LOCKLIST` and `MAXLOCKS` database configuration parameters. The default values of these parameters may not be big enough (`LOCKLIST`: 10 pages, `MAXLOCKS`: 10%) and cause excessive lock escalations.

To determine the lock list size, estimate the following numbers:

- Average number of locks per application
- Maximum number of active applications

If you have no idea of the average number of locks per application, execute an application and monitor the number of held locks at the application level using the Snapshot Monitor. To get the number of locks held by a particular application, execute the Snapshot Monitor as in the following example:

```
GET SNAPSHOT FOR LOCKS FOR APPLICATION AGENTID 15
```

In this example, 15 is the application handle number, which you can obtain using the `LIST APPLICATIONS` command.

See Chapter 5, “Monitoring tools and utilities” on page 119 for detailed information about the database system monitor including the Snapshot Monitor and Event Monitor.

For the maximum number of active applications, you can use the value of MAXAPPLS database configuration parameter.

Then perform the following steps to determine the size of your lock list:

- Calculate a lower and an upper bound for the size of your lock list using the following formula:

  ```
  (Average number of locks per application * 36 * maxappls) / 4096
  (Average number of locks per application * 72 * maxappls) / 4096
  ```

  In the formula above, 36 is the number of bytes required for each lock against an object that has an existing lock, and 72 is the number of bytes required for the first lock against an object.

- Estimate the amount of concurrency you will have against your data, and based on your expectations, choose an initial value for the LOCKLIST parameter that falls between the upper and lower bounds that you have calculated.

You may want to increase LOCKLIST if MAXAPPLS is increased, or if the applications being run perform infrequent commits.

When setting MAXLOCKS, you should consider the size of the lock list (LOCKLIST) and how many locks you will allow an application to hold before a lock escalation occurs. If you will allow any application to hold twice the average number of locks, the value of the MAXLOCKS would be calculated as following:

```
100 * (average number of locks per application * 2 * 72 bytes per locks)
/ (lock list * 4096 bytes)
```

You can increase MAXLOCKS if few applications run concurrently, since there will not be a lot of contention for the lock list space in this situation.

Once you have set the LOCKLIST and MAXLOCKS database configuration parameters, you can use the Snapshot Monitor and Event Monitor to validate or adjust the value of the values of these parameters. Here are the monitor elements which you should be interested in:

- Maximum number of locks held by a given transaction
- Total lock list memory in use
- Number of lock escalations that have occurred

You can check the maximum number of locks held by a given transaction using the Event Monitor. You need to create an event monitor to get transaction events to get this information. This information can help you to determine if your application is approaching the maximum number of locks available to it, as defined by the LOCKLIST and MAXLOCKS database configuration parameters. In order to perform this validation, you will have to sample several applications. Note that the monitor information is provided at a transaction level, not an application level.

To check the total lock list memory in use, you should use the Snapshot Monitor at the database level. If you notice that the monitored value is getting closer to the lock list size, consider increasing the value of the LOCKLIST parameter. Note that the LOCKLIST configuration parameter is allocated in pages of 4K bytes each, while this monitored value is in bytes.

To check the number of lock escalations that have occurred, you can use the Snapshot Monitor at database level. If you observe many lock escalations, you should increase the value of the LOCKLIST and/or the MAXLOCKS parameters.

See Chapter 5, “Monitoring tools and utilities” on page 119 for detailed information about the Event Monitor and Snapshot Monitor.

7.6.8 Lock wait behavior

What happens if one application requests to update a row that is already locked with an exclusive (X) lock? The application requesting the update will simply wait until the exclusive lock is released by the other application.

To ensure that the waiting application can continue without needed to wait indefinitely, the LOCKTIMEOUT database configuration parameter can be set to define the length of the time-out period. The value is specified in seconds. By default, the lock time-out is disabled (set to a value of -1). This means the waiting application will not receive a time-out and wait indefinitely.

7.6.8.1 Statement level rollback

If a transaction waits for a lock longer than the time the LOCKTIMEOUT parameter specifies, the entire transaction will be rolled back by default. You can make this roll back due to time-out at statement level by setting the DB2 registry variable DB2LOCK_TO_RB=STATEMENT by the following command:

```
db2set DB2LOCK_TO_RB=STATEMENT
```

This command should be executed by the instance owner, and you need to stop/start the database manager to make this change effective. If you set DB2LOCK_TO_RB=STATEMENT, lock time-outs cause only the current statement to be rolled back.

7.6.8.2 Deadlock behavior

In DB2, contention for locks by processes using the database can result in a deadlock situation.

A deadlock may occur in the following manner:

- A locks record 1.
- B locks record 5.
- A attempts to lock record 5, but waits since B already holds a lock on this record.
- B then tries to lock record 1, but waits since A already holds a lock on this record.

In this situation, both A and B will wait indefinitely for each other's locks, until an external event causes one or both of them to roll back.

DB2 uses a background process, called the deadlock detector, to check for deadlocks. The process is activated periodically as determined by the `DLCHKTIME` parameter in the database configuration file. When activated, it checks the lock system for deadlocks.

When the deadlock detector finds a deadlock situation, one of the deadlocked applications will receive an error code and the current unit of work for that application will be rolled back automatically by DB2. When the rollback is complete, the locks held by this chosen application are released, thereby allowing other applications to continue.

To monitor deadlocks, you can use the Snapshot Monitor at the database level as well as the application level.

Since eliminating unnecessary locks minimizes the possibility of deadlocks, tips we have discussed this section are also applicable to avoid deadlocks, therefore:

- Issue `COMMIT` statements at the right frequency.
- Specify the `FOR FETCH ONLY` clause in the `SELECT` statement.
- Specify the `FOR UPDATE` clause in the `SELECT` statement.
- Choose the appropriate isolation level.
- Eliminate next key locks by setting `DB2_RR_TO_RS=YES` if acceptable.
- Release read locks using the `WITH RELEASE` option of the `CLOSE CURSOR` statement if acceptable.

- Avoid lock escalations impacting concurrency by tuning `LOCKLIST` and `MAXLOCKS` parameter.

Chapter 8. Tuning database utilities

Data and information are the lifeblood of every organization. With the increasing presence of the Internet, e-business solutions, business automation, and data warehousing demands, continuous data availability is now a requirement. When business processes are affected by an outage, it is very critical to be able to define both a recovery point and recovery time objectives to specify how fast we need to recover data and how recent the data will be. Whenever an outage occurs, we must recover quickly to a point at which usable data is available.

Speedy recovery depends on regular backups and fail-safe procedures for converting backup data to production data. Although it seems strange to consider backup/restore procedures and performance at the same time, as very large databases are becoming more prevalent, the performance of a backup/restore solution becomes a very critical variable in ensuring data availability. In this section, we detail the various options that have some quantitative measure of the improvement of BACKUP, EXPORT, LOAD, IMPORT and RESTORE utilities.

8.1 BACKUP DATABASE utility

Backups should be done whenever there is a logical breakpoint in the business cycle, for example, at the end of a business day. To make a backup copy of the database, use the `BACKUP DATABASE` utility from the command line processor, from the Control Center, or from application programs with the administrative API, `sqlubkp`. This utility automatically establishes a connection to the specified database. If a connection to the specified database already exists, it will be used for the backup operation, and the connection will be terminated at the completion of the backup. You need to consider the following before running the `BACKUP DATABASE` utility:

- Start the database manager (`DB2START`) before running the `BACKUP DATABASE` command or API. For the Control Center, explicit starting of the database manager is not required.
- You must have SYSADM, SYSCTRL, or SYSMAINT authority to issue the `BACKUP DATABASE` utility.
- While using the command, API, or task under the Control Center, specify the database alias name, not the database name.
- Do not back up a database that is not in a usable state, except for a database in the backup pending state.

- Re-execute the BACKUP DATABASE utility if a system crash occurs during a critical stage of backup.
- Ensure that the concurrent backup processes do not target to the same tape.
- Be aware that the BACKUP DATABASE utility provides a concurrency control for multiple processes which are making backup copies of different databases. This control feature keeps the backup target device open until the entire backup process has ended.

In the art of performance tuning, the objective is usually to keep the output device busy (for example, tape). As long as the buffers are being filled as quickly as they can be written to the device, then we have done well.

For additional information, please see *Recovering a Database* in the *Administration Guide: Implementation*, SC09-2944.

8.1.1 Command options

Although we have many command line options for BACKUP DATABASE, here we will consider just a few of the command options that can provide a considerable amount of performance improvement while using the backup utility.

8.1.1.1 TO dir/dev

The target directory and devices must reside on the database server. This parameter is repeated to specify the target directories and devices that the backup image will span across. If more than one target is specified (for example bkup_dir1, bkup_dir2, and bkup_dir3), then bkup_dir1 is opened first, and the media header and special files are placed along with the configuration file, table space table, and history file. All the remaining targets are opened, and are then used in parallel during backup, thereby increasing the performance and reducing the time for overall backup operation.

In order to take a backup quickly for large databases, when you have only a few tape devices, it is preferable to back up the database on multiple target directories on multiple disks or on different mounted volumes. Then this can be moved to tape at the earliest convenient time.

```
BACKUP DATABASE sample TO /bkup_dir1,/bkup_dir2,/bkup_dir3
```

8.1.1.2 BUFFER buffer-size

This value is used as the buffer allocation size in pages (4 KB) when building the backup image. For a buffer size of zero, the value of the database manager configuration parameter `BACKBUFSZ` will be used as the buffer allocation size. When backing up a database, the data is first copied to an internal buffer. Data is then written from this buffer to the backup media when the buffer is full. Tuning `BACKBUFSZ` can help improve the performance of the backup utility as well as minimizing the impact on the performance of other concurrent database operations.

We recommend setting the buffer size to a multiple of the extent size. If multiple table spaces have different extent sizes, the buffer size value should be a multiple of the largest extent size. See 3.3.8.3, “Extent size” on page 65.

For a user on most versions of Linux, using DB2's default buffer sizes for backup and restore to a SCSI tape device results in the error “SQL2025N, reason code 75”. To prevent the overflow of Linux internal SCSI buffers, use the following formula to determine the appropriate buffer size:

```
Bufferpages <= ST_MAX_BUFFERS * ST_BUFFER_BLOCKS / 4
```

Here, `Bufferpages` is the value of either `BACKBUFSZ` or `RESTBUFSZ` if you do not explicitly specify the `USING BUFFER` parameter.

`ST_MAX_BUFFERS` and `ST_BUFFER_BLOCKS` are defined in the Linux kernel under the `drivers/scsi` directory.

For additional information on the configurable database manager configuration parameters please refer to the *Administration Guide: Performance*, SC09-2945.

8.1.1.3 OPEN n SESSIONS and WITH n BUFFERS option

While using the `OPEN n SESSIONS` option along with TSM (formerly ADSM), specifying `OPEN x SESSIONS` will establish x connections to the TSM servers. This `OPEN n SESSIONS` command has no effect when backing up to tape, disk, or other local devices. Each of the sessions will wait for data buffers to become available. For this reason, we require at least x+1 backup buffers to be allocated. The default for `num-buffers` is 2.

When creating a backup to multiple locations using multiple sessions of TSM or multiple local devices, a larger number of buffers may be used to improve performance.

```
BACKUP DATABASE sample USE TSM OPEN 2 SESSIONS WITH 4 BUFFERS
```

The number of buffers to allocate should be:

```
Number of Buffers = #sessions +#parallelism +2
```

and

```
(num-buffers * buffer-size) < UTIL_HEAP_SZ
```

If you use a variable block size with your tape devices, ensure that the DB2 buffer size is either less than or equal to the maximum variable block size for which the device is configured. After choosing the configuration, make sure to test both backup and restore with those settings. Otherwise, the backup will succeed, but the resulting image may not be guaranteed to be recoverable.

8.1.1.4 PARALLELISM n

Using this parameter, we can dramatically reduce the time required to complete the backup. This parameter defines the number (n) of processes that are started to read data from the database. Each process is assigned to backup a specific table space. When it completes backing up the table space, it requests another. Each process will be assigned a table space to complete, therefore, to get better performance, let this value be less than the number of table spaces; since setting up the value higher than the number of table spaces does not show any significant difference in performance. However, each process requires both memory and CPU overhead; for a heavily loaded system, you should leave this parameter at its default value of 1.

Note

On BACKUP, parallelism must be less than number of table spaces.

8.1.2 Configuration parameters

Along with the command options, we can also tune a few database manager configuration parameters to gain considerable impact on performance. The customer’s specific needs and environment will determine the tuning effort of these parameters.

8.1.2.1 Utility heap size (util_heap_sz)

This is the maximum database shared memory that can be used simultaneously by the `BACKUP`, `RESTORE` and `LOAD` utilities and during the load recovery process. It is recommended to use the default values unless your utilities run out of space; in that case, increase this value. You cannot run these utilities concurrently when the value is set too low because of memory constraints.

During offline backup, you can increase the `UTIL_HEAP_SZ` and the backup buffer size (using the `BUFFER` option or `BACKBUFSZ` parameter) and subsequently reduce the buffer pool size (`BUFFSIZE`) and others which are not relevant to backup, thereby obtaining the maximum backup performance. Ensure that the original values are restored after the backup is over.

Tips

It is recommended to have the following values set:

```
util_heap_sz > 2* (backbufsiz + restbufsiz
```

8.1.2.2 DB2_DISABLE_FLUSH_LOG

Though this registry variable does not result in a direct performance impact, it provides an option to disable the inclusion of the last active log file in any on-line backups. When an on-line backup completes, the last active log file is truncated, closed, and archived as part of the backup. This behavior gives us a complete backup, including all of the logs required for the restoring of that backup.

For a detailed description, please refer to 3.5.7, “Flushing logs during on-line backup” on page 81.

8.1.3 DB2CHKBP

It is always important to ensure that the backup images are restorable. The `db2chkbp` utility helps to test the integrity of a backup image and determine whether or not it can be restored. If the backup was created using multiple sessions, `db2ckbkp` can examine all of the files at the same time. Users are responsible for ensuring that the session with sequence number 001 is the first file specified. This utility also verifies backup images on tape.

Please follow these recommendations:

- Include the plan for backing up of databases during the design of the system itself to ensure high performance.
- Have more than one source for taking backup (for example, tape drive) to facilitate the use of multiple targets during backup, thereby improving backup performance.
- Test the backup and tapes regularly.
- Perform a disaster recovery test at least once per year.

8.1.4 DB2ADUTL

The utility `db2adutl` allows you to query, extract, verify and delete backups, logs, and load copy images saved using Tivoli Storage Manager (formarly ADSM). On UNIX based systems, this utility is located in the `$INSTHOME/sqllib/misc` directory. Please see the *Command Reference*, SC09-2951 for additional information. This utility also helps to ensure that the backup images are in restorable form.

8.1.5 DB2LOOK

It is recommended to have all the DDL statements to reproduce the database objects of a production database. The `db2look` tool extracts the required DDL and also generates the required `UPDATE` statements used to replicate the current statistics on the objects in test database, updates database configuration, and updates database manager configuration parameters and `db2set` statements, so that the registry variables and configuration parameter settings on the test database match those of the production database. With this tool, we can make a test system containing a subset of the production system's data, where access plans are similar to those that would be used on the production system. Thus, without affecting the production system, we can try to test performance measurements in the test system.

8.1.6 RUNSTATS, REORGCHK, and REORG

It is recommended to use the `RUNSTATS` command to collect current statistics on tables and indexes when frequent insert, update and delete activity has occurred, or when new indexes have been created since the last time the `RUNSTATS` command was executed. Also, if the `REORGCHK` command indicates that a reorganization should be done on the table data, you need to perform `REORG`, then `RUNSTATS`, to provide the optimizer with current statistics. Even though running statistics takes a considerable amount of time, checking and performing `RUNSTATS`, `REORG`, and `REBIND` of applications before taking backup will help you to get better performance before and after restore.

In the `REORGCHK` output, `CLUSTERRATIO` indicates the percentage of table data that is organized according to an index. If this value is less than 80, the table needs to be reorganized according to the most used index. When a table has multiple indexes, you have to specify the most important index for reorganizing the data. For organization of the indexes, `100*NPAGES/FPAGES` value has to be determined from the `REORGCHK`. If this value is less than 80, the indexes should be dropped and re-created. DB2 UDB provides the facility to perform an automatic index reorganization online without the need to force the connected users and applications (see 4.2.2.1, "On-line index reorg" on page 108).

8.2 EXPORT utility

The EXPORT utility exports data from a database to one of several external file formats. The data to be exported can be specified by a SELECT statement, or by providing hierarchical information for typed tables (which we do not discuss in this book). For additional information on typed tables, please see the *SQL Reference*, SC09-2974. If the output file name already exists, then the export process overwrites the contents of the file; the information is not appended. You should have SYSADM or DBADM authority, or the CONTROL or SELECT privilege, on each participating table or view.

The exported data can then be imported or loaded into another DB2 database, using the IMPORT or the LOAD utility, respectively; or it can be imported into another application (for example, a spreadsheet)

The following information is required when exporting data:

- A SELECT statement must be provided for exporting the required data.
- The path and name of the file that will store the exported data should be given.
- The format of the data in the input file (for example, IXF, WSF, or DEL) is needed.
- A message file name must be specified.
- Sub-table traverse order within the hierarchy is needed when exporting typed tables. When specifying the order, sub-tables must be traversed in the pre-order fashion. When exporting typed tables, you must specify the target sub-table name with (optionally) a WHERE clause. The export utility uses this information, along with the traverse order, to generate and execute the required SELECT statement. See *Moving Data between Typed Tables* under *Data Movement Utilities Guide and Reference*, SC09-2955 for more information.
- While exporting LOB data, it is better to provide one or more paths to store LOB files so that when the file space is exhausted in the first path, the second path can be used for storage and, so on.

Note

While exporting data in WSF format with BIGINT or DECIMAL data, only values that fall within the range of type DOUBLE can be exported accurately. Even though those values that do not fall within this range are exported, importing or loading these values back may result in incorrect data, depending on the operating system

Also, check for restrictions for EXPORT in *Data Movement Utilities Guide and Reference*, SC09-2955.

The file formats that can be exported are:

- Delimited ASCII format (DEL)
- Work sheet format (WSF)
- PC Integrated exchange format (IXF)

Note

When using IXF, Version 7 tables with a schema name greater than 8 characters cannot be imported or exported with pre-Version 7 code because of truncation occurs

For additional information on file formats, please see the *Data Movement Utilities Guide and Reference,* SC09-2955.

The EXPORT utility is an embedded SQL application and does SQL fetches internally. This means that all optimizations on those SQL operations apply for export too (for example, a large buffer pool size). Changes on the server side that improve fetch performance can have a larger impact.

8.2.0.1 METHOD N column-name

Specifying target table column names while exporting as WSF or IXF, will have considerable performance impact during IMPORT or LOAD process. If this parameter is not specified during EXPORT, the column names in the table are used. Usually, when you specify column name or column position during IMPORT or LOAD operation, additional CPU time is required for parsing the data. The LOAD utility does not support WSF format.

8.3 IMPORT utility

The `IMPORT` utility inserts data from an external file with a supported file format into a table, hierarchy, or updatable view. A faster alternative is `LOAD`; however, the `LOAD` utility does not support loading data at the hierarchy level. If the table or view receiving the imported data already contains data, you can either replace or append to the existing data. After importing of data the `RUNSTATS` utility should be run to get updated statistics. Some of the advantages of using `IMPORT` are:

- Concurrent access to the table space
- Fire triggers
- Clustered index support
- WSF file format support
- Support to import into tables and views
- Superior granularity for commits
- Concurrent maintenance of constraints, automated summary tables, referential integrity
- The `IMPORT` utility is an embedded SQL application and does SQL inserts internally. This means that all optimizations on those SQL operations apply to import too (for example, a large buffer pool size). Changes to the server side that improve insert performance can have a much larger impact.

Note

Do not set the agent pool size (NUM_POOLAGENTS) to zero. If the agent pool size is zero, an agent process will be generated and terminated repeatedly for each row, and will impact on the import performance significantly.

These authorizations are required to use the `IMPORT` utility:

- To create a new table, you must have SYSADM authority, DBADM authority, or CREATETAB privilege for the database.
- To replace data in an existing table or view, you must have SYSADM authority, DBADM authority, or CONTROL privilege for the table or view.
- To append data to an existing table or view, you must have `SELECT` and `INSERT` privileges for the table or view.

For further authorization information, see *Administration Guide - Implementation*, SC09-2944.

Before invoking the IMPORT utility, you must be connected to (or able to perform an implicit connect to) the database into which the data will be imported. Since the utility will acquire a table level exclusive lock, you should complete all transactions and release all locks on the target table. You can invoke the IMPORT utility either from Command Line Processor (CLP) or from Control Center.

The following information is required when importing data:

- The path and the name of the input file.
- The name or alias of the target table or view.
- The format of the data in the input file (such as IXF, WSF, DEL, or ASC)
- The type of operation, whether input data has to be inserted, or updated or replaced.
- A message file name, if utility is invoked using an API call.
- The method to use: column location, column name, or relative column position.
- The number of rows to insert before committing the changes (COMMITCOUNT).
- The number of file records to skip, if applicable (RESTARTCOUNT).
- The names of the columns within the table or view into which the data is to be inserted.
- When working with *typed tables*, you may also need to provide:
  - The method or order by which to progress through all of the structured types. Ordering is important when moving data between table hierarchies, because it determines where the data is moved in relation to other data.
  - The sub-table list. For details, see *Moving Data Between Typed Tables* in D*ata Movement Utilities Guide and Reference*, SC09-2955.

Also, check for *Restrictions and limitations for Import* in the *Data Movement Utilities Guide and Reference*, SC09-2955.

The IMPORT utility creates two temporary files, one for data, and the other for messages generated. These temporary files are located in the `tmp` subdirectory under the `sqllib` directory or the directory indicated by the DB2INSTPROF registry variable.

If you receive an error while writing or opening data on the server, ensure the following conditions:

- The directory exists.
- There is sufficient disk space for the files.
- The instance owner has write permission in the directory.

By default, automatic COMMITs are not performed for `INSERT` or the `INSERT_UPDATE` option. They are performed only when `COMMITCOUNT` is not zero.

When you import a file to a remote database, internally a stored procedure is called to perform the import on the server. This stored procedure will not be called when:

- The application and database code pages are different.
- The import file is a multiple-part PC/IXF file.
- The method used is either column name or relative column position.
- The target column list provided is longer than 4 KB.
- An OS/2 or DOS client is importing a file from diskette.
- The `LOBS FROM` clause or the `LOBSINFILE` modifier is specified.
- The `NULL INDICATORS` clause is specified for ASC files.

8.3.1 METHOD options

Different method options parameters can be used to import different input file types, such as ASC, IXF and DEL.

- **L** — Specifies the start and end column numbers from which to import data. This method can only be used with ASC files, and is the only valid option for that type.
- **N** — Specifies the names of the columns to be imported. This method can only be used with IXF files.
- **P** — Specifies the indexes of the input data fields to be imported. This is numbered from 1. This method can be only be used with IXF or DEL files. But this is the only valid option for the DEL type files.

Specifying target table column names or a specific importing method (either L, N, or P) makes the importing to a remote database slower, due to additional tasks in parsing.

8.3.2 MODIFIED BY COMPOUND=x

If this option is used, then DB2 will insert one block at a time, rather than one row at a time. This can give you better performance results.

In this option, x is a number between 1 to 100 inclusive. IMPORT uses non-atomic compound SQL to insert the data, and x statements will be attempted each time.

```
CONNECT TO SAMPLE
IMPORT FROM myfile OF IXF MODIFIED BY COMPOUND=100 INSERT INTO mytable
```

The above IMPORT command will wait for the SQL return code about the insert results after inserting 100 records, instead of inserting each record.

If this modifier is specified, the transaction log must be large enough to accommodate the number of rows in the data file or number of rows specified by the COMMITCOUNT, if it is specified. It is recommended to use COMMITCOUNT option along with COMPOUND to avoid transaction log overflow.

This modifier is incompatible with INSERT_UPDATE mode, USEDEFAULTS, hierarchical tables or generated column modifiers, that includes INDENTITYMISSING, IDENTITYIGNORE, GENERATEDMISSING and GENERATEDIGNORE.

See the *Data Movement Utilities Guide and Reference*, SC09-2955 for more information.

8.3.3 COMMITCOUNT n

By default, with no COMMITCOUNT specified, an IMPORT will issue a commit at the end of a successful IMPORT. Requesting periodic COMMITs reduces the number of rows that are lost if a failure occurs during the import operation. It also prevents the DB2 logs from getting full when processing a large input file.

If an import operation is interrupted, a COMMITCOUNT was specified, the table is usable, and it will contain the rows that were loaded up to the last COMMIT, then you can restart the import operation, or accept the table as is. In case you want to restart the import operation, you can specify the number of file records to skip (equivalent to the number successfully imported and committed already). Though a smaller COMMITCOUNT improves concurrency and enables us easily to control and proceed IMPORT, it will provide additional CPU overhead.

8.3.4 Logging

During an insert operation, every row that is being inserted into a table will be written to the DB2 logs for recovery purpose. When you are doing a mass import, then it is necessary to have the correct number of primary and secondary logs, with appropriate log file size, to avoid getting a transaction log full error. A simple rule of thumb, used to calculate log file size, is 3 times the table size if there is no index and 5 times the table size if there is an index, and the total log file size limitation of 32 GB. That is, (LOGPRIMARY + LOGSECOND) * LOGFILSZ * 4 K should be less than 32 GB.

The log full condition is possible with both circular and archival logging. In the case of circular logging, if an open transaction holds the first log file, when other transactions use the last defined log file, they cannot continue by writing over the first. Similarly, archival logging cannot have more than the maximum number (LOGPRIMARY + LOGSECOND) of defined active log files. Until the first log file in the sequence of active log files is released, the next one cannot be created.

8.3.4.1 Log buffer size (LOGBUFSZ)

Buffering the log records will result in more efficient log file I/O, because the log records will be written to disk less frequently, and more log records will be written at each time. This parameter allows you to specify the amount of the database heap (defined by the DBHEAP parameter) to use as a buffer for log records before writing these records to disk. The log records are written to disk when one of the following occurs:

- A transaction commits or a group of transactions commit, as defined by the MINCOMMIT configuration parameter.
- The log buffer is full.
- As a result of other internal database manager event, like logger EDU (db2logger).
- When the log file is full.

If logging is the primary constraint during IMPORT, then do the following:

- Make sure that the application which does the import operation commits more often (decrease the COMMITCOUNT value so that it will commit more often).
- If applicable, perform a LOAD operation instead of an IMPORT, since it does less logging.
- Increase the number of secondary logs (LOGSECOND), since these will get allocated when DB2 needs them.

For more information, please see *Appendix B: Differences Between the Import and the Load Utility* in the *Data Movement Utilities Guide and Reference*, SC09-2955.

8.4 LOAD utility

The `LOAD` utility moves data into newly created tables or into tables that already contain data, extends existing indexes, and generate statistics. Data from the `EXPORT` utility can be loaded with the `LOAD` utility. Usually `LOAD` is used with large data sets and accepts both binary and character based data. Here you can also specify the number of rows to load, but loaded data information is not logged.

You should have `LOAD` authority to run the `LOAD` utility without the need for SYSADM or DBADM authority. This gives database administrators more granular control over the administration of their database. See the *Data Movement Utilities Guide and Reference*, SC09-2955 for more information. Since all load processes are owned by the instance owner, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the `LOAD` utility. Also, check for *Restrictions and Limitations* for `LOAD` in *Data Movement Utilities Guide and Reference,* SC09-2955.

The `LOAD` utility is faster than the `IMPORT` utility for bulk data, because it writes formatted pages directly into the database, while the import utility performs SQL inserts. The `LOAD` utility can take advantage of intra-partition parallelism and I/O parallelism. Loading data is a heavily CPU-intensive task. The `LOAD` utility takes advantage of multiple processors for tasks such as parsing and formatting data. Also, the `LOAD` utility can use parallel I/O servers to write the data to the containers in parallel.

The `LOAD` utility works in three phases:

1. Load — During this phase, data is loaded into the table. Index keys and table statistics are collected and partially sorted, if indexes are defined. Associated table spaces are in load-pending state from the beginning to the end of the build phase.
2. Build — During this phase, indexes are created based on the index keys collected during the load phase, and the sort is completed. Associated table spaces are in delete-pending state from the end of the build phase until the end of the delete phase.

3. Delete — During this phase, the rows that caused a unique key violation are removed from the table. These are placed in the exception table, and messages about the rejected rows are written to the message file.

Note

Since the LOAD utility does not check any constraints defined on the table (such as check constraints and referential constraints) except unique constraints, you need to execute a SET INTEGRITY statement after the loading to make the table usable. If all the load operations are performed in INSERT mode, the SET INTEGRITY statement will check the table for constraint violations incrementally, by checking only the appended portion of the table.

Using the LOAD utility is a very efficient way to insert data, for the following reasons:

- Data I/O is performed only once for all three phases.
- In case of index creation, LOAD uses a heavily optimized sort routine which is considerably faster than the generic sort routine of the DBMS.

Note

In Version 7, SORT BUFFER and temporary sorting directories for LOAD are now obsolete. LOAD uses a sort heap for sorting, and performs I/O in temporary table spaces.

Since the LOAD utility will lock the table spaces to which the table belongs, you should complete all transactions and release all locks by performing a COMMIT or ROLLBACK.

If a particular sequence is desired, the data has to be sorted before LOAD, since the LOAD utility will load the data in the same sequence that appeared in the input file.

The Version 7 LOAD utility can load data residing on a remotely connected client, by specifying the CLIENT parameter. Data residing on the server may be in the form of a file, tape, or named pipe. Data residing on a remotely connected client may be in the form of a fully qualified file or named pipe. This CLIENT option is ignored if the load operation is not being invoked from a remote client. Separate files containing LOB values, when the lobsinfile file type modifier is specified, should be located on the server. Also, if the CLIENT option is specified, the path must be fully qualified.

For additional information on the LOAD utility, see the *Command Reference*, SC09-2951. For new LOAD behavior and changes, see *Changes to Previous Load Behavior Introduced in Version 6 and Version 7*, under the *Data Movement Utilities Guide and Reference*, SC09-2955.

The following load parameters are now intelligently defaulted (self-tuning feature) at runtime. If you do not specify them, the loader will examine the system configuration and available resources, at runtime, and select an appropriate value for each. They are

- Data buffer
- Sort buffer
- CPU parallelism
- Disk parallelism

In making these decisions, the loader consider factors such as:

- Available memory remaining in the utility heap
- Number of on-line CPUs (SMP)
- Table design (whether it has indexes, or LONG data)
- Number of table space containers

8.4.1 Command options

Although there are many command line options for LOAD, we will consider just a few which will provide a considerable amount of performance improvement.

8.4.1.1 SAVECOUNT n

This parameter value represents the interval, in number of rows, between consistency points. During the LOAD process, after every *n* rows are loaded, a consistency point will be established. Table objects like long field, LOB, BLOB are stored in table spaces. I/O to each table space is performed in extents and the extent size is measured in 4K pages, specified during table space creation parameter. LOAD builds extents for each table object in the appropriate form, converts into page count, and rounded up to intervals of extents size and writes the resulting pages to disk.

If a very large number of rows is to be loaded, it is recommended to specify a large SAVECOUNT value, thereby reducing time for the synchronization activities performed to establish a consistency point. A LOAD RESTART operation will automatically continue from the last constancy point. The default value is 0, means no constancy points will be established.

8.4.1.2 COPY YES or NO

Specifying `COPY YES` (when forward recovery is enabled) will save a copy of the loaded data so that the table can be recovered. This reduces load performance, because all the loaded data is copied; however, you need to specify this option if you want the table to be recoverable (or you need to take an offline backup). Specifying multiple target devices or directories on different file systems, using `TO` device/directory, can reduce I/O time for the backup image, thereby a performance gain. Note that tape is not supported in OS/2, and copy to tapes in SCO UnixWare 7 is not supported.

`COPY NO` may reduce overall performance, because if the forward recovery is enabled, the table is placed in backup pending state, and the database, or selected table spaces, must be backed up before the table can be accessed. When forward recovery is enabled and you want the target table to be recoverable, specify `COPY YES`.

8.4.1.3 DATA BUFFER buffer-size

`DATA BUFFER` can reduce I/O waits when loading long fields or LOBs. This memory is allocated directly from the utility heap (`UTIL_HEAP_SZ`). If the value is not specified, an intelligent default (self-tuning future) is calculated by the loader by examining the system configuration, and available resources, at runtime. While specifying this value, ensure that this buffer be several extents in size. An extent is the unit of movement for data within DB2 UDB, and this size can be one or more 4KB pages. It is highly recommended not to specify this parameter, so that DB2 will allocate the best value.

8.4.1.4 CPU_PARALLELISM n

This parameter value specifies the number of processes that the load utility will spawn for parsing, converting, and formatting data records, while building the table data objects. If your machine has the capability, this parameter will exploit intra-partition parallelism and significantly improve load performance. This parameter can be used in non-SMP environments also, but performance benefit is not perceptible.

CPU parallelism does not lose data set order, that is, the data provided in a sequence a, b, c, d will be loaded in parallel, but will arrive on disk in the identical order a, b, c, d.

Although you can specify a maximum value of 30, the utility adjusts the value if there is insufficient memory. If the value of this parameter is zero, or has not been specified, the load utility uses an intelligent default value, usually based on the number of CPUs available, at run time. It is highly recommended not to specify this parameter, so that DB2 will allocate the best value.

Specifying a small value for the SAVECOUNT parameter causes the loader to perform many I/O operations to flush both data and table metadata. When CPU_PARALLELISM is greater than 1, the flushing operations are asynchronous, permitting the loader to exploit the CPU. When CPU_PARALLELISM is set to 1, the loader waits on I/O during consistency points.

Note

Parallelism is not supported in cases where tables include LOB or LONG VARCHAR data.

8.4.1.5 DISK_PARALLELISM n

This parameter values specifies the number of processes used by the load utility to write data records to disk. Use this parameter to exploit available containers when loading data, and get significant improvement in load performance. The maximum number allowed is the greater of four times the CPU_PARALLELISM value (used by the LOAD utility), or 50. If the value of this parameter is zero, or has not been specified, the load utility uses intelligent default value based on the number of table space containers and the characteristics of the table. It is highly recommended not to specify this parameter, so the LOAD utility will use an intelligent default value.

8.4.1.6 MODIFIED BY filetype-mod

There are few valid file type modifiers that will improve the performance of the LOAD utility significantly. They are as follows:

FASTPARSE

The modifier FASTPARSE will reduce the syntax checking and data checking that is performed on user-supplied column values, and enhance performance. Though this option performs sufficient data checking to prevent a segmentation violation or trap, this option should be used only when the data being loaded is known to be valid and architecturally correct, since FASTPARSE assumes that your data is clean and yields a performance gain when CPU-bound.

This increases performance more on ASCII data than on PC/IXF data, since IXF is a binary format, and FASTPARSE affects parsing and conversion from ASCII to internal forms. Data that is in correct form will be loaded correctly and care must be taken to use this modifier with clean data only. If you have confidence that your data is clean, use this option for additional performance gains, especially with character data files (DEL and ASC).

ANYORDER

Use this file type modifier to suspend the preservation of source order in the data being loaded, and gain performance on SMP systems. This modifier is used along with the CPU_PARALLELISM parameter. If the data to be loaded is presorted, ANYORDER may corrupt the presorted order, and the benefits of presorting will be lost for subsequent queries. If CPU_PARALLELISM is 1, then this option is ignored.

Note

This option is not supported if SAVECOUNT > 0, since the recovery process after a consistent point requires that data to be loaded in sequence. When SAVECOUNT > 0, avoid using the ROWCOUNT option along with ANYORDER, because which row(s) get loaded cannot be guaranteed

BINARYNUMERICS

Use this type of file-type modifier only when loading positional numeric ASC (non-delimited ASCII) data into fixed length records specified by the RECLEN option (NOEOFCHAR is assumed). When numeric (INT, REAL, FLOAT, and not DECIMAL) data be in binary form and not in the character representation, avoids costly conversions, thereby providing a performance gain. Use IXF or positional ASCII with BINARYNUMERICS whenever possible, since binary data loads much faster than text data.

While using BINARYNUMERICS, the following rules apply:

- Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running.
- Data lengths MUST match their target column definitions.
- FLOATs must be in IEEE Floating Point format.
- No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT.

Note

NULLs cannot be present in the data for columns affected by this modifier. Blanks are interpreted as binary value when this modifier is used.

PACKEDDECIMAL
DB2 loads packed decimal data directly, since the BINARYNUMERICS modifier does not include the DECIMAL field type. Use this file type modifier to improve performance when loading positional numeric ASC data into fixed-length records specified by RECLEN option.

8.4.1.7 STATISTICS YES

Use this operation to collect data distribution and index statistics for the table and for any existing indexes more efficiently than by the RUNSTATS utility, with compromise on LOAD performance. This option is supported only if the load operation is in REPLACE mode.

Apart from tuning LOAD, it is recommended to tune job stream also. Thus, when you consider the end-to-end time to get the table populated and ready for access, it is recommended to combine index creation, and RUNSTATS with your load. With a single pass through the data, you can create an index, collect statistics, and make the table available for queries (single I/O against many operations).

Once the statistics are updated, applications can use new access paths to the table data from the latest statistics. However, new access paths to the table can be created by rebinding the application packages using the REBIND command.

When distribution statistics are being gathered or when loading data into large tables, it is recommended to have a larger value for the database configuration parameter Statistics Heap Size (STAT_HEAP_SZ). For more information about STAT_HEAP_SZ, see *Database Parameters,* in the *Adminstration Guide: Performance*, SC09-2945.

8.4.1.8 NONRECOVERABLE

This parameter can improve the load performance when forward recovery is enabled (LOGRETAIN or USEREXIT is ON). Use this parameter if you do not need to recover load transactions against a table. Many users do not want certain loaded tables to be recoverable because the data is transient (replaced frequently). When LOGRETAIN or USEREXIT is ON, performing a load operation without the COPY YES option will bring the table space into backup pending state; however, using the NONRECOVERABLE option completes the load operation without leaving the table spaces in backup pending state. This load is non-recoverable, although the database has LOGRETAIN and USEREXIT ON, and a copy of the loaded data does not have to be made during the load operation.

Rollforward will skip non-recoverable `LOAD` and ignore subsequent transactions against the table; therefore, further actions against this table are ignored. After rollforward recovery, a table may only be replaced (using `IMPORT` or `LOAD`) or dropped. Load performance is enhanced, because no additional operation is performed, apart from movement of data into the table.

8.4.1.9 Indexing mode

This option specifies whether the `LOAD` utility is used to rebuild indexes or to extend them incrementally. Supported modes include `AUTOSELECT`, `REBUILD`, `INCREMENTAL`, and `DEFERRED`.

- `REBUILD` performs best when appending a large percentage of new data (for example, >50%)
- `INCREMENTAL` performs best when appending small volumes of data (for example, <10%)
- The default behavior is `AUTOSELECT`. This mode uses an internal runtime performance costing model to select between `REBUILD` and `INCREMENTAL` modes. `AUTOSELECT` determines best behavior in the grey zone of 10 to 50.
- `DEFERRED` is best when performing multiple `LOAD INSERT` operations or when you would rather not maintain indexes during load.

For information about when the above modes can be used, see *Chapter 3: LOAD* in the *Data Movement Utilities Guide and Reference*, SC09-2955.

8.4.2 Considerations for creating an index

When tuning index creation performance, the amount of memory dedicated to the sorting of index keys during a `LOAD` operation is controlled by the `SORTHEAP` database configuration parameter. Tuning sort has a direct impact on load performance when the target table has indexes. Therefore, tuning sort also tunes aspects of load.

8.4.2.1 SORTHEAP

This parameter defines the maximum number of private memory pages used for private sorts, or the maximum number of shared memory pages to be used for shared sorts. If the sort is a private sort (this is what `LOAD` uses), then this parameter affects the agent private memory. If the sort is a shared sort, then this parameter affects the database global memory. This sort heap is the area where data is sorted.

This parameter indicates the number of 4K pages of memory we will try to allocate when sorting index keys for each index defined on the table. If the table has 4 indexes, then we will try to allocate 4 * SORTHEAP pages in total. In order to control the total amount of memory allocated, another configuration parameter, sort heap threshold (SHEAPTHRES), is also significant.

When the total amount of instance-wide memory allocated for private sorts exceeds SHEAPTHRES, then subsequent sorts will try to allocate fewer than SORTHEAP pages of memory. SHEAPTHRES should be set to some reasonable multiple of SORTHEAP.

Sort overflows (or sort-spill) are sorts that ran out of sort heap and required disk space for temporary storage. That is, the data is divided for several sort runs and stored in a temporary table space that will be merged later. These sorts are not efficient, and when this value is consistently high for a number of intervals, then it may be necessary to increase the SORTHEAP configuration parameter. If the sort spill occurs even after increasing the SORTHEAP parameter, make the buffer pool for temporary table space large enough to minimize the amount of disk I/O.

To examine these configuration parameters, issue these commands:

```
DB2 GET DBM CFG | grep -i sheapthres
DB2 GET DBM CFG FOR <dbname> | grep -i sortheap
```

To set the sheapthres value to 32768 (or whatever value you desire):

```
DB2 UPDATE DBM CFG USING sheapthres 32768
```

To set the sortheap value to 8192; try this, using 8192 of 4K pages for sorting each index:

```
DB2 UPDATE DB CFG FOR <dbname> USING sortheap 8192
```

After changing these values, it is best to disconnect from the database, do a DB2STOP and DB2START and then continue, otherwise, the new values might not be used.

In terms of performance, the higher the value of SORTHEAP, the less likely a sort-spill will be and you will get better performance due to reduced I/O.

In V6 and later, sort-spills go to temp tables (there are no temporary files as in V5). In order to improve I/O during spill processing, it is important to do the following:

1. Make sure that the buffer pool for your temporary tables pace is a reasonable size. If it is too small, this could adversely affect performance.

2. Make sure that the temporary tables pace is defined using multiple-containers, each defined on a separate device. This will help to increase I/O parallelism during spill processing.

While increasing SORTHEAP, see whether SHEAPTHRES also needs to be increased.

8.4.2.2 SHEAPTHRES

This performance variable sort heap threshold (sheapthres), is used to tune the database manager configuration. This controls the total amount of memory for sorting across the entire instance for all sorts. On the first connect we calculate the total size of the database shared memory set, which includes BUFFERPOOLS, UTILHEAPSZ, SHEAPTHRES, PACKAGE CACHE, and LOCKLIST. The database shared memory set cannot be dynamically modified, that is, you must terminate all connections to the database and reconnect.

Please see 6.4.1, “Sorting methods” on page 201 for more information.

Note

Whenever you adjust the sorting configuration parameters, use an operating system monitor to track any changes in system paging. If the post threshold sorts are high, you can increase sheapthres and/or decrease sortheap.

The performance of LOAD can be improved by installing high performance sorting libraries from third party vendors to create indexes during load operation. A few of the vendor extensions include:

- LOAD utilities for DB2 from BMC, and Platinum.
- Host driven data loading product, Optiloader from LSI, provides host centric control of loading jobs, and additional functionality, such as UPSERT.
- Pluggable sort products to enhance index creation available from SyncSort
- PatrolDB, Unload utility from BMC.

8.4.3 Load query

You can use the LOAD QUERY command to check the status of a load operation during processing. When you see very poor load performance, you should execute this command and check the current status of the processing load. For example, if the input data does not match the target columns (data type, data length, and so on), the LOAD utility will generate numerous warning messages which will degrade performance.

8.4.4 Loading data into a table with clustered index

Newly inserted rows often cannot be placed in the same physical sequence as the logical sequence defined by the index. This can be avoided by using a clustered index. The cluster factor of a clustered index is maintained or improved dynamically as data is inserted into the associated table, by attempting to insert new rows physically close to the rows for which the key values of this index are in the same range. Only one clustered index may exist for a table, so the CLUSTER option during CREATE INDEX may not be specified if it was used in the definition of any existing index on the table. Also, a clustered index may not be created on a table that is defined to use append mode, and is disallowed if a nickname is specified. Here are some advantages in using a clustered index:

- Use a CLUSTER INDEX to optimize queries that retrieve multiple records in INDEX order.
- When a CLUSTER INDEX is defined, use ALTER TABLE..PCTFREE nn before LOAD or REORG. This will leave nn% free space on the table's data pages after LOAD and REORG.

Since the LOAD utility preserves the order of the input file and does not consider the cluster factor, when load data into a table with a clustered index, you should:

- Use the IMPORT utility instead of the LOAD utility.
- If you are performing the LOAD in REPLACE mode (or loading into a new table), pre-sort the input file and execute the LOAD command without the ANYORDER option.
- If you are performing the LOAD in APPEND mode, execute the REORG TABLE command after the load.

8.5 RESTORE DATABASE utility

Backing up the database allows recovery of the database either fully or partially in case, a database become unusable due to failure of hardware or software (or both), or an operating system failure. The backup and recovery type depends on how the logging mode has been set. You can use this command only when your database has been previously backed up using the BACKUP DATABASE command.

RESTORE DATABASE can be invoked by using the Command Line Processor (CLP), the Control Center, or administrative API, sqlurestore. It rebuilds a damaged or corrupted database that has been backed up using the BACKUP DATABASE command.

You can restore backup images which are produced by previous versions (V2.x or higher) of DB2, in case migration is required, it will be issued automatically at the end of restore. If you are using the Control Center, then you cannot restore backups that were taken in previous versions of DB2.

The database to which you restore the data may be the same one as the data was originally backed up from, or it may be different (in addition to being able to restore to a new database). If, at the time of backup operation, the database was enabled for roll-forward recovery, the database can be brought to the state that it was in, prior to the occurrence of the damage or corruption, by issuing ROLLFORWARD DATABASE after successful execution of RESTORE DATABASE.

You can select a type at the time of restore, from three types:

1. A full restore of everything from the backup.
2. A restore of only the recovery history file.
3. A subset of the table spaces in the backup.

A database seed is a unique identifier of a database that remains constant for the life of the database. This seed is assigned by the database manager when the database is first created. The seed is unchanged following a restore of a backup, even if the backup has a different database seed. DB2 always uses the seed from the backup. You need to consider the following before running the RESTORE command:

- The database manager must be started before restoring a database.
- You must have SYSADM, SYSCTRL, or SYSMAINT authority to restore to an existing database from a full database backup. To restore to a new database, you must have SYSADM or SYSCTRL authority.
- To restore to an existing database, you require a database connection. To restore to a new database, you require instance attachment. To restore to a new remote database, it is necessary to first attach to the instance where the new database will reside.
- The type of restore: full restore, restore recovery history file, or table space.
- When you use Tivoli Storage Manager (TSM) utility, any restrictions of that utility should also be considered.

- If there is more than one backup on the source media, then the `TAKEN AT` parameter has to be used to specify the timestamp when the backup was taken. If you use TSM and do not specify the `TAKEN AT` parameter, TSM retrieves the latest backup copy.
- If a system failure occurs during any stage of restoring a database, you cannot connect to the database until you reuse the `RESTORE` command and successfully complete the restore.
- If the code page of the database being restored does not match a code page available to an application; or, if the database manager does not support code page conversions from the database code page to a code page that is available to an application; then the restored database will not be usable.
- Directory and file containers are automatically created, if they do not exist. No redirection is necessary unless the containers are inaccessible for some reason.

For additional information, please see *Chapter 8: Recovering a Database* in the *Administration Guide: Implementation*, SC09-2944.

8.5.1 Command options

Although there are many command line options for `RESTORE`, we will consider just a few of the command options that can provide a considerable amount of performance improvement while using the `RESTORE` utility.

8.5.1.1 OPEN n SESSIONS and WITH n BUFFERS option

While using the `OPEN num-sessions SESSIONS` option along with TSM (formally ADSM), the `OPEN x sessions` will establish `x` connections to the TSM servers. Thus, `num-sessions` signifies the number of I/O sessions to be used with TSM or the other vendor product. The default for `num-buffers` is 2. However, a larger number of buffers may be used when multiple sources are being read or when parallelism parameter is increased; this will considerably reduce the amount of time required to do a restore.

Tips

The number of buffers to allocate should be `#sessions +#parallelism +2` and `(num-buffers * buffer-size) < util_heap_sz`.

Parallelism is discussed in the following section.

8.5.1.2 PARALLELISM n

Specifies the number of buffer manipulators to be spawned during the restore process. The default value is 1. For restore, you can increase the number higher but not higher than the number of TSM Client sessions you have started. As well as you will need at least the parallelism number of buffers allocated.

Note

On restore, parallelism can be as high as the # sessions used.

8.5.1.3 BUFFER buffer-size

You may specify the number of pages (4K) to use for each restore buffer when you invoke the `RESTORE` command. When restoring a database, the data is first copied from the backup media to an internal buffer. Data is then written from this buffer to the target database media when the buffer is full. Using large restore buffers helps to improve the performance of the restore utility.

We recommend setting the restore buffer size to a multiple of the extent size; however, the value you specify must be equal or a multiple of the backup buffer size that you specified when the backup image was taken. Otherwise, the smallest acceptable size of the buffer (4KB) is used for the restore buffer. If you do not specify the number of pages, each buffer will be allocated based on the database manager configuration parameter `RESTBUFSZ`.

You can also specify the number of the restore buffer. When restring from multiple locations using multiple sessions of TSM or multiple local devices, a larger number of buffers may be used to improve performance.

The number of buffers to allocate should be:

```
Number of Buffers = #sessions +#parallelism +2
```

and

```
(num-buffers * buffer-size) < UTIL_HEAP_SZ
```

Note

When you set the number and size of the restore buffer, try to use more buffers of smaller size rather than a few large buffers.

If variable blocking is used for the tape device, be sure to use a BUFFER size equal to or less than the maximum blocking factor of the tape device. For example, for a tape drive with maximum blocking on 64 KB, the buffer size should not exceed 16 (4KB) pages.

8.5.2 Configuration parameters

Along with the command options, we can also tune the RESTORE related database manager configuration parameters to gain considerable impact on performance. The customer's specific needs and environment will determine the tuning effort on these parameters.

8.5.2.1 Utility heap size (util_heap_sz)

This is the maximum database shared memory that can be used simultaneously by the BACKUP, RESTORE and LOAD utilities and during the load recovery process. It is recommended to use the default values unless your utilities run out of space; in that case, increase this value. We cannot run these utilities concurrently when the value is set too low because of memory constraints.

Tips

It is recommended to have the following values set:

```
util_heap_sz > 2* (backbufsiz + restbufsiz)
```

8.5.2.2 Restore buffer size (restbufsz)

If the BUFFER size is not explicitly specified when issuing the RESTORE command, this configuration parameter value is used. This specifies the size of buffer used when restoring the database.

Chapter 9. Resolving performance problems

When an application is expected to have a better response time than is being obtained, the process of diagnosing the problem starts. If the workload on a system increases, the throughput of the system may be maintained, but individual application response time may be degraded.

In this chapter, we discuss various conditions that may cause performance problems and explain how to resolve them. Finally, we show a sample case to improve the performance using the Explain and the db2batch tools.

9.1 Identifying the cause

As in any other problem-determination technique, the first step requires the database administrator to be able to reproduce or identify the problem. Some symptoms may be sporadic; others may be permanent.

To identify the performance problem, ask your users, who should be able to tell you which operations of an application are suffering from performance problems. This will help you determine exactly when to monitor your system in order to isolate the problem. Alternatively, you may want to perform a general analysis of your system to try to isolate possible bottlenecks during peak periods.

After the problem is reproduced or identified, it will fall into one of two groups:

1. Problems affecting one application or a group of applications:

   Problems that affect a single application or a group of applications can be further subdivided into two categories:

   - Applications that have had a good performance history in a development or testing environment, but do not perform as expected when working against production databases. Working against low volumes of data may hide problems. Some of the non-detected problems may be those associated with casting, lack of indexes, joins, sorts, access plans, isolation levels, or size of the answer set.
   - Applications whose behavior is erratic. These applications may usually have good response times, but under certain conditions, their response times are very degraded. These applications may have concurrence-related problems: deadlocks, waits, and so on.

2. Problems affecting all applications:

   Problems that affect all applications usually appear when changes are made to data loads, the number of users, the operating system, or the database configuration parameters. The cause of these types of problems is usually found in the following areas:

   - Configuration parameters (sorts, buffer pool, logs, lock list). Sometimes, this is not caused by a modification in the parameter, but by the environment itself. Bigger tables that require a larger sort heap or the updating of more rows in a table may require a larger log buffer. Also, more users exhausting the lock list and provoking concurrence problems can cause problems that affect all database applications.
   - Operating system problems, such as I/O contention or excessive paging.
   - Network problems, if the clients or applications are remote.
   - Data access problems, where access plans may be obsolete, statistics may not have been updated, or packages may not be rebound.

9.2 Application problems

When you find performance problems affecting one or a group of applications, the first thing you should do is to check whether most of the elapsed time is spent in the applications, or in DB2. If most of the time is being spent in the applications, it is not worthwhile to tune DB2 before the application problem is eliminated.

For CLI, ODBC, JDBC or SQLJ applications, the CLI/ODBC/JDBC trace facility (see Chapter 5, “Monitoring tools and utilities” on page 119) is a good tool you can use to perform this step.

Once you have found that most of the elapsed time is spent in DB2, then you should analyze the SQL statements issued in the application. Pick up the most time-consuming SQL statements from the dynamic SQL Snapshot Monitor (for dynamic SQL) or from the Statements Event Monitor (for both dynamic and static SQL), then analyze these SQL statements using the following two basic procedures:

- Explaining the statements
- Monitoring the application/database

9.2.1 Explaining the statements

This can be done through Visual Explain or any of the Explain tools. The access plan will show the work that the database manager needs to perform to retrieve the answer set. This should be compared to the access plan that you expected.

The access plan provides information not only about the work that the database manager has to perform, but also how the work will be done.

Run custom queries to get information about your Explain plans and check the following:

1. Search for SORT, or GROUP BY operators on sets of columns and base tables that occur frequently, which could be beneficial as an index or summary tables.
2. Search for expensive operations, such as large or spilling sorts, high buffer usage, or high table queue usage.
3. Search for expensive plans to further examine for database optimizations.
4. Search for common predicates that could form potential start/stop keys for an index.
5. Check for missed index opportunities.
6. Look for any other better join opportunities.
7. Search for poor predicate selectivities due to insufficient statistics.
8. Search for FETCH used because an index could use INCLUDE columns.
9. Search for I/O increase during SORT (using vmstat or iostat).

If you are not satisfied with the access plan being shown, you can obtain different access plans for different levels of optimization (see 7.3.2.3, “Tune optimization level” on page 248) without needing to execute the statement.

For dynamic SQL statements, different levels of optimization will deliver different access plans, but they also will show different times needed to prepare the statement. The balance of the time required to prepare the statement and the time required to execute it will yield better performance for the statement.

The steps of an access plan and its description are presented in *Description of db2expln and dynexpln Outpu*t in the *Administration Guide: Performance*, SC09-2945.

9.2.2 Monitoring the application/database

The application can be monitored through the performance monitor, taking a snapshot of the application, or defining an event monitor for the statements or the transaction. For dynamic SQL statements, the `db2batch` tool also will provide snapshots (for the application, database, and instance, if the appropriate level of detail is selected) and will measure the response time of the statement/statements. Application monitoring will collect values for data elements that can point to performance problems.

Data elements whose values are collected when monitoring the application include: deadlocks; lock escalations; lock waits and lock wait time; index and data reads and writes; number of sorts and sort time; and the package cache hit ratio.

9.3 Database configuration problems

Many of the database configuration problems are detected through SQLCODEs or SQLSTATEs that are returned to applications. If the error handler routines are well written, they should present the error to the end user. These errors should lead to the cause of the problem and will facilitate the problem-determination process.

When a database configuration problem is suspected, but there is no certainty about the conflicting parameter or parameters, the database should be monitored. This monitoring can be achieved using the snapshot monitor, the event monitor or a combination of these. When planning to monitor the database environment, you need to choose a significative period of time for monitoring to take place; such as a 60-minute interval during peak hours. Take time to examine the output collected from the monitoring tools, and check for data elements that can point to specific configuration problems. These data elements can be high-water marks, overflows or rejections.

Define a method for resolving problems, and use it consistently. Here are some guidelines for establishing a problem-determination method:

- Choose the monitoring tool.
- Define the period of time and the environment in which the database will be monitored.
- Start monitoring the database
- Obtain the results of the monitoring tool.

- Based on the results of the monitored data elements, select the parameter to be modified. Modify only the selected configuration parameter. Remember to restart the database.
- Reestablish the original monitoring environment, if possible.
- Monitor again, obtaining the results of the data elements with the new value of the configuration parameter.
- Compare the results obtained.
- If results are not positive, reestablish the configuration parameter to its old value.

When a database is monitored, a large set of values for data elements may be collected. Some values may point directly to the cause of a problem, but that is not always the case. Table 12 shows some of the data elements that can be collected, their related configuration parameters and the problems caused by incorrect configuration values. The table is only an example and is not intended as a complete listing for problem determination. Many of the data elements collected will relate directly to configuration parameters. When using the performance monitor, is it possible through the online help to identify which configuration parameter relates to the data elements.

Table 12. Data elements and configuration problems

Data element	**Configuration parameters**	**Probable causes**
Catalog cache overflows	catalogcache_sz	Catalog cache is too small.
Catalog cache heap full	catalogcache_sz dbheap	The dbheap is too small compared to catalog cache. The dbheap may fill up if the size of the buffer pool is increased.
Post threshold sorts	sortheap sheapthres	Sort heap threshold is too small.
Sort overflows	sortheap	If too many, applications are requiring a bigger sort heap.
Lock time outs	locklist	If too many, there is a concurrence problem
Lock waits	locklist	If too many, there is a concurrence problem

Data element	Configuration parameters	Probable causes
Package Cache overflow	pckcachesz	Insufficient memory causes error SQLCODE -973 and catalog table lock contention.
Deadlocks detected		If too many, there is a concurrence problem.
Lock escalations	locklist maxlocks	Lock list is too small, applications monopolizing the lock list.
Dirty page threshold cleaner triggers	buffpage chngpgs_thresh	If too many, the buffer pool is running out of free pages too often. The buffer pool size is too small, or chngpgs_thresh is too low.

The best way to evaluate if there are “too many” overflows, time-outs, or waits is to compare results to previous results or a similar environment. When possible, results should be compared to those obtained when the database did not have a performance problem.

Concurrence problems occur only when more than one application is accessing the database. They may point to an application problem related to the isolation levels being used. They also may point to an insufficient size of the lock list memory area. Notice that a snapshot monitor for locks may be taken. This can provide valuable information to determine the cause of the problem.

Data elements can be grouped to obtain ratios. Ratios are presented by the performance monitor or can be calculated by the database administrator. The manual, *System Monitor Guide and Reference*, SC09-2956 contains a description of all the data elements for which information can be collected and how to calculate ratios using the different elements. An example of these ratios and their relationship to configuration parameters is shown in Table 13.

Table 13. Ratios and configuration problems

Ratio	Parameters	Probable causes
Buffer Pool Hit Ratio	Specified buffer pool size (CREATE BUFFERPOOL or ALETER BUFFERPOOL)	If too low, prefetchers not working, buffer pool too small.
Buffer Pool Index Hit Ratio	Specified buffer pool size (CREATE BUFFERPOOL or ALTER BUFFERPOOL)	If too low, prefetchers not working, buffer pool too small.
Catalog cache hit ratio	catalogcache_sz	If too low, catalog cache too small.
Percentage of sorts that overflow	sortheap	If significant, sort heap too small.

9.4 Data access problems

Data access is the most probable cause of performance problems that can affect all applications. To avoid these problems, the following steps can be taken:

- The database administrator should keep the statistics updated and should periodically bind/rebind for static SQL applications.
- The database administrator should reorganize tables periodically.

To check if reorganizations are required, DB2 provides the `REORGCHK` utility. When a performance problem is suspected, and data access is suspected to be the cause, the database tables should be checked. If reorganizations show a low clustering ratio for the clustering index of a table, then that table should be reorganized. If clustered indexes are not being used, a big performance gain could be obtained by reorganizing tables according to the most accessed index.

Operating system tools indicating I/O contention may point to problems with the physical design of the database, such as placement of containers across physical drives, or containers allocated to table spaces (see Chapter 3, “Data storage management for performance” on page 31).

9.5 Case study

We now present a case study that uses the Explain facility and the db2batch tool to analyze a query and resolve its performance problems.

This study is based on the database schema provided by the Transaction Processing Performance Council (TPC). In this case study, we use two tables, the `LINEITEM` table and the `ORDERS` table. The column names and their data types are shown in Table 14 and Table 15.

Table 14. Lineitem table

Column names	Data types	Column length (bytes)
L_ORDERKEY	FLOAT	8
L_PARTKEY	INTEGER	4
L_SUPPKEY	INTEGER	4
L_LINENUMBER	INTEGER	4
L_QUANTITY	FLOAT	8
L_EXTENDEDPRICE	FLOAT	8
L_DISCOUNT	FLOAT	8
L_TAX	FLOAT	8
L_RETURNFLAG	CHAR	1
L_LINESTATUS	CHAR	1
L_SHIPDATE	DATE	4
L_COMMITDATE	DATE	4
L_RECEIPTDATE	DATE	4
L_SHIPINSTRUCT	CHAR	25
L_SHIPMODE	CHAR	10
L_COMMENT	VARCHAR	44

Table 15. Order table

Column names	Data types	Column length (bytes)
O_ORDERKEY	FLOAT	8
O_CUSTKEY	INTEGER	4
O_ORDERSTATUS	CHAR	1
O_TOTALPRICE	FLOAT	8
O_ORDERDATE	DATE	4
O_ORDERPRIORITY	CHAR	15
O_CLERK	CHAR	15
O_SHIPPRIORITY	INTEGER	4
O_COMMENT	CHAR	79

The LINEITEM table has 6001215 rows and the ORDERS table has 1500000 rows.

Using the following select statement, we show how we could improve the performance:

```
SELECT COUNT(*) FROM lineitem, orders
  WHERE l_extendedprice=o_totalprice
    AND o_orderdate >= '01/01/1998'
  GROUP BY o_custkey
```

In this case study, we set 10 seconds of the elapsed time as the performance tuning goal.

To execute the select statement, we used the db2batch tool with PERF_DETAIL level 5 (see 5.4.5, “The db2batch utility” on page 173) so that we could get a snapshot for the database manager, database, and this query statement. We also used the dynamic Explain tool (dynexpln) with the -g option to get the section information and the access plan graph of the query.

The page size of the table space for the LINEITEM and ORDERS was 8 KB, therefore, the page size of the buffer pool for this table space was also 8 KB. This buffer pool was called TPCDATABP in this case study.

The SYSCATSPACE used the buffer pool IBMDEFAULTBP . The page size of IBMDEFAULTBP was 4 KB.

The database was deactivated/activated between each invocation of the query to eliminate the benefit of pre-loading the buffer pools so that each test run was consistent. In a production database, significant performance gains would be made if the buffer pools already contained the tables' or indexes' data pages.

9.5.1 Non-tuned environment

We started the case study with the default settings in the database manager and database configuration parameters.

Note

Normally you should not use the default values. You should use the Configure Performance Wizard to obtain recommended values for your environment instead

The following is the package information from the dynamic Explain output:

```
Package Name = TETSUR3.DYNEXPLN
   Prep Date = 2000/07/07
   Prep Time = 12:55:57

   Bind Timestamp = 2000-07-07-12.55.57.610196

   Isolation Level          = Cursor Stability
   Blocking                 = Block Unambiguous Cursors
   Query Optimization Class = 5

   Partition Parallel       = No
   Intra-Partition Parallel = No
   Function Path            = "SYSIBM", "SYSFUN", "TETSUR3"
```

Note that the partition parallelism is NO, which is the default value.

Here is the access plan graph section information from the Explain output:

```
       RETURN
       (   1)
         |
       GRPBY
       (   2)
         |
       TBSCAN
       (   3)
         |
        SORT
       (   4)
         |
       MSJOIN        <=This shows the type of Join strategy
       (   5)
      /      \
  TBSCAN    TBSCAN
  (   6)    (  10)
    |         |
   SORT      SORT    <= Sort before join
  (   7)    (  11)
    |         |
  TBSCAN    TBSCAN   <= Table Scans are performed
  (   8)    (  12)
    |         |
 Table:    Table:
 DB2INST1  DB2INST1
 ORDERS    LINEITEM
```

Here is the section information from the Explain output:

```
Estimated Cost        = 968724
Estimated Cardinality = 99996

Access Table Name = DB2INST1.ORDERS  ID = 4,9
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
|  Sargable Predicate(s)
|  |  #Predicates = 1
|  Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 2
|  |  #Sort Key Columns = 1
|  |  |  Key 1: O_TOTALPRICE (Ascending)
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 133417
|  |  |  Row Width = 16
|  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 2
|  Relation Scan
|  |  Prefetch: Eligible
Merge Join
```

```
|  Access Table Name = DB2INST1.LINEITEM  ID = 4,7
|  |  #Columns = 1
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Temp Table  ID = t2
|  |  |  #Columns = 1
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: L_EXTENDEDPRICE (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 6001215
|  |  |  |  Row Width = 12
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 1
|  |  Relation Scan
|  |  |  Prefetch: Eligible
Insert Into Sorted Temp Table  ID = t3
|  #Columns = 2
|  #Sort Key Columns = 1
|  |  Key 1: (Ascending)
|  Sortheap Allocation Parameters:
|  |  #Rows     = 99996
|  |  Row Width = 12
|  Piped
|  Buffered Partial Aggregation
Access Temp Table  ID = t3
|  #Columns = 2
|  Relation Scan
|  |  Prefetch: Eligible
|  Final Predicate Aggregation
|  |  Group By
|  |  Column Function(s)
Final Aggregation Completion
|  Group By
|  Column Function(s)
Return Data to Application
|  #Columns = 1

End of section
```

In the example of Explain output shown above, you can identify the following:

- Estimated Cost is high (= 968724).
- Relation Scan has been selected for the LINEITEM and ORDERS table.
- Sort operations are performed before joining the ORDERS table and LINEITEM table, and the sort key is the O_TOTALPRICE column for the ORDERS table, the L_EXTENDEDPRICE column for the LINEITEM table.

Here is the partial output from the db2batch tool:

```
Elapsed Time is:          216.471    seconds
:
:
Sort heap allocated                          = 0
Total sorts                                  = 3
Total sort time (ms)                         = 130326
Sort overflows                               = 2
Active sorts                                 = 0
:
:
Bufferpool Name                               = IBMDEFAULTBP
:
Buffer pool data logical reads                = 209
Buffer pool data physical reads               = 27
Buffer pool data writes                       = 3
Buffer pool index logical reads               = 45
Buffer pool index physical reads              = 7
Buffer pool index writes                      = 0
Total buffer pool read time (ms)              = 18
Total buffer pool write time (ms)             = 79
:
:
Bufferpool Name                               = TPCDATABP
:
Buffer pool data logical reads                = 221219
Buffer pool data physical reads               = 168149
Buffer pool data writes                       = 47618
Buffer pool index logical reads               = 0
Buffer pool index physical reads              = 0
Buffer pool index writes                      = 0
Total buffer pool read time (ms)              = 83965
Total buffer pool write time (ms)             = 13507
```

From the db2batch output, you should notice the following:

- The elapsed time to execute the query (= 216.471)
- Total sort time in ms (= 130326)
- Sort overflows (= 2)
- Physical reads, both data and index (168149)

9.5.2 Tune configuration parameters

As emphasized throughout this book, you should run the Configure Performance Wizard to get the recommended values for the database manager/database configuration parameters.

Here is the list of recommended values we obtained from the Configure Performance Wizard for our case study environment:

```
APP_CTL_HEAP_SZ 160
BUFFPAGE        95487
CATALOGCACHE_SZ 326
CHNGPGS_THRESH  40
DBHEAP          1552
LOCKLIST        162
LOGBUFSZ        8
LOGFILSIZ       1000
LOGPRIMARY      3
LOGSECOND       10
MAXAPPLS        40
MAXLOCKS        50
MINCOMMIT       1
NUM_IOCLEANERS  1
NUM_IOSERVERS   5
PCKCACHESZ      160
SOFTMAX         100
SORTHEAP        4570
STMTHEAP        4096
DFT_DEGREE      ANY
DFT_PREFETCH_SZ 64
UTIL_HEAP_SZ    63991
SHEAPTHRES         27425
INTRA_PARALLEL     ON
MAX_QUERYDEGREE    4
MAXAGENTS          200
NUM_POOLAGENTS     -1
NUM_INITAGENTS     0
FCM_NUM_BUFFERS    160
FCM_NUM_RQB        128
ALTER BUFFERPOOL IBMDEFAULTBP SIZE 95487
ALTER BUFFERPOOL TPCDATABP SIZE 48243
```

After applying the recommended values shown above, we run the dynamic Explain tool. The following is the access plan graph:

```
              RETURN
              (   1)
                |
              GRPBY
              (   2)
                |
               LMTQ
              (   3)
                |
              TBSCAN
              (   4)
                |
               SORT
              (   5)
                |
              MSJOIN
              (   6)
             /      \
     TBSCAN    TBSCAN
     (   7)    (  11)
       |          |
      SORT       SORT
     (   8)    (  12)
       |          |
     TBSCAN    TBSCAN
     (   9)    (  13)
       |          |
    Table:     Table:
    DB2INST1   DB2INST1
    ORDERS     LINEITEM
```

Here is the section information from the Explain output:

```
Intra-Partition Parallelism Degree = 4

Estimated Cost        = 247174
Estimated Cardinality = 99996

Process Using 4 Subagents
|  Access Table Name = DB2INST1.ORDERS  ID = 4,9
|  |  #Columns = 3
|  |  Parallel Scan
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Sargable Predicate(s)
|  |  |  #Predicates = 1
|  |  Insert Into Sorted Shared Temp Table  ID = t1
|  |  |  #Columns = 2
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: O_TOTALPRICE (Ascending)
|  |  |  Use Partitioned Sort
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 133417
|  |  |  |  Row Width = 16
|  |  |  Piped
|  Sorted Shared Temp Table Completion  ID = t1
|  Access Temp Table  ID = t1
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Merge Join
|  |  Access Table Name = DB2INST1.LINEITEM  ID = 4,7
|  |  |  #Columns = 1
|  |  |  Parallel Scan
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  |  |  Insert Into Sorted Shared Temp Table  ID = t2
|  |  |  |  #Columns = 1
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  Key 1: L_EXTENDEDPRICE (Ascending)
|  |  |  |  Use Partitioned Sort
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 6001215
|  |  |  |  |  Row Width = 12
|  |  |  |  Piped
|  |  Sorted Shared Temp Table Completion  ID = t2
|  |  Access Temp Table  ID = t2
|  |  |  #Columns = 1
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  Insert Into Sorted Temp Table  ID = t3
|  |  #Columns = 2
|  |  #Sort Key Columns = 1
|  |  |  Key 1: (Ascending)
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 99996
|  |  |  Row Width = 12
|  |  Piped
|  |  Buffered Partial Aggregation
|  Access Temp Table  ID = t3
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 2
|  Output Sorted
|  |  #Key Columns = 1
|  |  |  Key 1: (Ascending)
Final Aggregation
|  Group By
|  Column Function(s)
Return Data to Application
|  #Columns = 1

End of section
```

In the Explain output above, you should notice that:

- Intra Parallel has been chosen, the query degree is 4.
- The estimated cost for the access plan has been reduced (=247174).
- Relation Scan (four agents perform this scan in parallel) has been selected for the `LINEITEM` and `ORDERS` table.
- Sort operations are performed before joining the `ORDERS` table and `LINEITEM` table, and the sort key is the `O_TOTALPRICE` column for the `ORDERS` table, the `L_EXTENDEDPRICE` column for the `LINEITEM` table.

Here is the partial output from the db2batch:

```
Elapsed Time is:            133.157    seconds
:
:
Sort heap allocated                        = 0
Total sorts                                    = 6
Total sort time (ms)                           = 120255
Sort overflows                                 = 2
Active sorts                               = 0
:
Bufferpool Name                             = IBMDEFAULTBP
:
:
Buffer pool data logical reads              = 223
Buffer pool data physical reads             = 25
Buffer pool data writes                        = 2
Buffer pool index logical reads                = 45
Buffer pool index physical reads               = 7
Buffer pool index writes                       = 0
Total buffer pool read time (ms)               = 2
Total buffer pool write time (ms)              = 64
:
:
Bufferpool Name                                = TPCDATABP
:
Buffer pool data logical reads                 = 229685
Buffer pool data physical reads                = 121194
Buffer pool data writes                        = 0
Buffer pool index logical reads                = 0
Buffer pool index physical reads               = 0
Buffer pool index writes                       = 0
Total buffer pool read time (ms)               = 68282
Total buffer pool write time (ms)              = 0
```

As you can see in this output, the performance was improved (from 216.471 sec. to 133.157 sec.); however, actually we saw that excessive activities to the paging space were occurring during the query was processed. The following is a vmstats report at that time:

```
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 268029   110   0   0   0   3   22   0 107   83  23  1  0 97  1
 1  0 271706   119   0   2 538 740 13649   0 650  549 520 18 24 36 22
 1  1 273093   125   0  36 277 314  368   0 572  540 477 12 21 28 39
 1  1 273191   127   0  58  72  78   93   0 726  652 829 30  7 42 21
 1  1 274158     0   0  33 168 210  296   0 647  601 687 20 15 37 29
 1  1 275092   127   0 101 262 305  450   0 680  416 659 23  6 47 24
 2  1 275092   122   0   8   7   9   18   0 825  768 1043 47  3 31 19
 2  1 275092   123   0   0   4   5   10   0 829  775 1061 50  3 30 17
```

Excessive activities to the paging space normally cause I/O bottleneck and impact on the overall system performance. One of the common causes of this situation is setting too big buffer pools (or other memory areas).

In our case study, we set the recommended values for each buffer pool size as well as the database manager and the database configuration parameters. Probably the Configure Performance Wizard had suggested too big values for some parameters and therefore this excessive paging activity was occurred.

We checked each recommended values and found that the size of the buffer pool, IBMDEFAULTBP was 95487 pages. We created TPCDDATABP buffer pool for the table space which the ORDER and LINEITEM table belonged to and only the system catalog tables used IBMDEFAULTBP. Apparently 95487 pages was too big size for the IBMDEFAULTBP. We decided to reduce the size to 5000 and executed the query again.

The access plan was not changed by this modification though, reducing the size of IBMDEFAULTBP improved the performance (from 133.157 sec. to 111.762 sec.) as the following db2batch output (partial) shows:

```
Elapsed Time is:          111.762    seconds
:
:
Sort heap allocated                           = 0
Total sorts                                   = 6
Total sort time (ms)                          = 109099
Sort overflows                                = 2
Active sorts                                  = 0
:
:
Bufferpool Name                                = IBMDEFAULTBP
:
Buffer pool data logical reads                 = 221
Buffer pool data physical reads                = 25
Buffer pool data writes                        = 1
Buffer pool index logical reads                = 45
Buffer pool index physical reads               = 7
Buffer pool index writes                       = 0
Total buffer pool read time (ms)               = 2
Total buffer pool write time (ms)              = 23
:
:
Bufferpool Name                                = TPCDATABP
:
Buffer pool data logical reads                 = 229689
Buffer pool data physical reads                = 121194
Buffer pool data writes                        = 0
Buffer pool index logical reads                = 0
Buffer pool index physical reads               = 0
Buffer pool index writes                       = 0
Total buffer pool read time (ms)               = 52119
Total buffer pool write time (ms)              = 0
:
```

In this db2batch output, you should notice that:

- The total sort time was very big (109099 ms).
- Two of the sort operations needed to use the temporary space (sort overflows).
- Lots of physical reads occurred (121194)

To deal with them, you can:

- Create indexes to avoid sort operations
- Increase the sort heap size to avoid overflowed sorts
- Increase the buffer pool size to increase the buffer pool hit ratio (reduce buffer pool data physical reads)

As you can see in the db2batch output, the total sort time was very big (109099 ms). Thus, we decided take the first option, which is creating new indexes to avoid heavy sort operations.

9.5.3 Add a new index

The Explain output shows that two sort operations were needed before joining the `ORDERS` table and the `LINEITEM` table so that a merge join can be performed. As the sort heap allocation parameters in the Explain output shows, the sort operation for the `LINEITEM` table is more costly than for the `ORDERS` table. To eliminate this sort operation, we created an index called `L_EP` on the join key, the `L_EXTENDEDPRICE` column. We executed the following statement:

```
CREATE INDEX L_EP on LINEITEM(L_EXTENDEDPRICE)
```

After creating indexes, you must not forget to execute a `RUNSTATS` command.

```
RUNSTATS ON TABLE db2inst1.orders AND DETAILED INDEXES ALL
```

Then we executed the dynamic Explain tool again and saw whether the different access plan would be taken.

Here is the access plan graph:

```
        RETURN
        (   1)
          |
        GRPBY
        (   2)
          |
         LMTQ
        (   3)
```

```
          |
        TBSCAN
        (   4)
          |
         SORT
        (   5)
          |
        NLJOIN          <= Nested Loop Join is performed
        (   6)
       /      \
 TBSCAN        IXSCAN   <= Index Scan is performed
 (   7)        (   6)
   |          /      \
  SORT   Index:    Table:
 (   8)  DB2INST1  DB2INST1
   |     L_EP      LINEITEM
 TBSCAN
 (   9)
   |
Table:
DB2INST1
ORDERS
```

Here is the section information from the Explain output:

```
Intra-Partition Parallelism Degree = 4

Estimated Cost        = 55598
Estimated Cardinality = 99996

Process Using 4 Subagents
|  Access Table Name = DB2INST1.ORDERS  ID = 4,9
|  |  #Columns = 3
|  |  Parallel Scan
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Sargable Predicate(s)
|  |  |  #Predicates = 1
|  |  Insert Into Sorted Temp Table  ID = t1
|  |  |  #Columns = 2
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: O_TOTALPRICE (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 133417
|  |  |  |  Row Width = 16
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t1
|  Access Temp Table  ID = t1
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Nested Loop Join
|  |  Access Table Name = DB2INST1.LINEITEM  ID = 4,7
|  |  |  #Columns = 1
|  |  |  Index Scan:  Name = DB2INST1.L_EP  ID = 1
|  |  |  |  Index Columns:
|  |  |  |  |  1: L_EXTENDEDPRICE (Ascending)
|  |  |  |  #Key Columns = 1
|  |  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  |  Stop Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  Index-Only Access
|  |  |  |  Index Prefetch: Eligible 5238
|  |  |  |  |  Insert Into Sorted Shared Temp Table  ID = t2
|  |  |  |  |  |  #Columns = 2
|  |  |  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  |  |  Key 1: (Ascending)
|  |  |  |  |  |  Use Round-Robin Sort
|  |  |  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  |  |  #Rows     = 99996
|  |  |  |  |  |  |  Row Width = 12
|  |  |  |  |  |  Piped
|  |  |  |  |  |  Buffered Partial Aggregation
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  Sorted Shared Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 2
|  Output Sorted
|  |  #Key Columns = 1
|  |  |  Key 1: (Ascending)
Final Aggregation
|  Group By
|  Column Function(s)
Return Data to Application
```

```
|  #Columns = 1

End of section
```

In this Explain output, you should notice that:

- The estimated cost was reduced (from 247174 to 55598).
- An index scan using the new index was chosen as the access method to the `LINEITEM` table.
- The sort operation for the `LINEITEM` table was no longer required.
- Nested loop join was selected.
- A table scan was used for the `ORDERS` table.

Here is the output (partial) from the db2batch:

```
Elapsed Time is:            15.217     seconds
:
:
Sort heap allocated                        = 0
Total sorts                                        = 4
Total sort time (ms)                               = 2106
Sort overflows                                     = 0
Active sorts                               = 0
:
:
Bufferpool Name                             = IBMDEFAULTBP
:
Buffer pool data logical reads              = 190
Buffer pool data physical reads             = 24
Buffer pool data writes                     = 1
Buffer pool index logical reads             = 45
Buffer pool index physical reads            = 7
Buffer pool index writes                    = 0
Total buffer pool read time (ms)            = 2
Total buffer pool write time (ms)           = 22
:
:
Bufferpool Name                             = TPCDATABP
:
Buffer pool data logical reads              = 32915
Buffer pool data physical reads             = 21840
Buffer pool data writes                     = 0
Buffer pool index logical reads             = 400971
Buffer pool index physical reads            = 5256
Buffer pool index writes                    = 0
Total buffer pool read time (ms)            = 13793
Total buffer pool write time (ms)           = 0
```

The performance was improved significantly (from 111.762 sec. to 15.217 sec.); however, it did not meet our goal yet. The goal of this performance tuning was 10 seconds of the elapsed time.

From the Explain and db2batch tools output, you can see that:

- Many buffer pool physical reads were required
- Entire rows of the ORDERS table were scanned

To deal with such problems, you can:

- Increase the buffer pool size to reduce the buffer pool physical reads
- Create an index on the ORDERS table so that an index scan can be chosen

We decided to take the first option.

9.5.4 Increase buffer pool size

We increased the size of the buffer pool TPCDATABP from 48243 pages (that was what the Configure Performance Wizard had recommended) to 90000 pages, and execute the query again.

Here is the access plan graph:

```
              RETURN
              (   1)
                |
              GRPBY
              (   2)
                |
               LMTQ
              (   3)
                |
              TBSCAN
              (   4)
                |
               SORT
              (   5)
                |
              NLJOIN
              (   6)
             /      \
      TBSCAN         IXSCAN
      (   7)         (   6)
        |           /      \
       SORT   Index:    Table:
      (   8)  DB2INST1  DB2INST1
        |     L_EP      LINEITEM
      TBSCAN
      (   9)
        |
     Table:
     DB2INST1
     ORDERS
```

Here is the section information from the Explain output:

```
Intra-Partition Parallelism Degree = 4

Estimated Cost        = 55598
Estimated Cardinality = 99996

Process Using 4 Subagents
|  Access Table Name = DB2INST1.ORDERS  ID = 4,9
|  |  #Columns = 3
|  |  Parallel Scan
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Sargable Predicate(s)
|  |  |  #Predicates = 1
|  |  Insert Into Sorted Temp Table  ID = t1
|  |  |  #Columns = 2
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: O_TOTALPRICE (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 133417
|  |  |  |  Row Width = 16
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t1
|  Access Temp Table  ID = t1
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Nested Loop Join
|  |  Access Table Name = DB2INST1.LINEITEM  ID = 4,7
|  |  |  #Columns = 1
|  |  |  Index Scan:  Name = DB2INST1.L_EP  ID = 1
|  |  |  |  Index Columns:
|  |  |  |  |  1: L_EXTENDEDPRICE (Ascending)
|  |  |  |  #Key Columns = 1
|  |  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  |  Stop Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  Index-Only Access
|  |  |  |  Index Prefetch: Eligible 5238
|  |  |  |  |  Insert Into Sorted Shared Temp Table  ID = t2
|  |  |  |  |  |  #Columns = 2
|  |  |  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  |  |  Key 1: (Ascending)
|  |  |  |  |  |  Use Round-Robin Sort
|  |  |  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  |  |  #Rows     = 99996
|  |  |  |  |  |  |  Row Width = 12
|  |  |  |  |  |  Piped
|  |  |  |  |  |  Buffered Partial Aggregation
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  Sorted Shared Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 2
|  Output Sorted
|  |  #Key Columns = 1
|  |  |  Key 1: (Ascending)
Final Aggregation
|  Group By
|  Column Function(s)
Return Data to Application
|  #Columns = 1

End of section
```

As you can see above, the access plan was not changed by increasing the buffer pool size.

Here is the db2batch output (partial):

```
Elapsed Time is:            15.232     seconds
:
:
Sort heap allocated                        = 0
Total sorts                                = 4
Total sort time (ms)                       = 2158
Sort overflows                             = 0
:
:
Bufferpool Name                             = IBMDEFAULTBP
:
Buffer pool data logical reads              = 203
Buffer pool data physical reads             = 24
Buffer pool data writes                     = 4
Buffer pool index logical reads             = 45
Buffer pool index physical reads            = 7
Buffer pool index writes                    = 0
Total buffer pool read time (ms)            = 2
Total buffer pool write time (ms)           = 102
:
:
Bufferpool Name                             = TCPDATABP
:
Buffer pool data logical reads                      = 32915
Buffer pool data physical reads                     = 21840
Buffer pool data writes                     = 0
Buffer pool index logical reads             = 400971
Buffer pool index physical reads            = 5256
Buffer pool index writes                    = 0
Total buffer pool read time (ms)            = 13766
Total buffer pool write time (ms)           = 0
```

In the db2batch output, you can see that the buffer pool data physical reads were not decreased and the performance was not improved.

We also tested the same query using the db2batch tool after increasing the size of the buffer pool TCPDATABP from 90000 pages to 100000 pages; however, that resulted in worse performance due to an excessive activity on the paging space.

In this case study, we executed the query after deactivating/activating the database so that the buffer pool could not have any cached data before running the query. That is why you see many buffer pool data physical reads in the db2batch output. When we executed the query without deactivating the database, the number of the buffer pool data physical reads was actually zero.

9.5.5 Add a new index

Increasing the buffer pool size did not help very much in this case study, so then we took the other approach. As you can see in the Explain output, a table scan and a sort operation for the `ORDERS` table was still performed.

In the query we used in this case study, you can see a predicate `"WHERE O_ORDERDATE >= '01/01'98'"`. We created an index called `O_OD` on the `O_ORDERDATE` column to satisfy the predicate:

```
CREATE INDEX O_OD on ORDERS (O_ORDERDATE)
```

You should not forget to execute a `RUNSTATS` command after creating new indexes.

```
RUNSTATS ON TABLE db2inst1.orders AND DETAILED INDEXES ALL
```

Here is the changed access plan graph:

```
              RETURN
              (   1)
                |
              GRPBY
              (   2)
                |
               LMTQ
              (   3)
                |
              TBSCAN
              (   4)
                |
               SORT
              (   5)
                |
              NLJOIN          <= Nested Loop Join is performed
              (   6)
             /      \
       TBSCAN        IXSCAN   <= Index Scan is performed for the LINEITEM table
       (   7)        (   6)
         |          /      \
        SORT   Index:    Table:
       (   8)  DB2INST1  DB2INST1
         |     L_EP      LINEITEM
       RIDSCN
       (  10)
         |
        SORT
       (  11)
         |
         IXSCAN   <= Index Scan is performed for the ORDERS table
         (  12)
        /      \
 Index:     Table:
 DB2INST1  DB2INST1
 O_OD      ORDERS
```

Here is the section information from the Explain output:

```
Intra-Partition Parallelism Degree = 4

Estimated Cost        = 52211
Estimated Cardinality = 104448

Process Using 4 Subagents
|  Access Table Name = DB2INST1.ORDERS  ID = 4,9
|  |  #Columns = 1
|  |  Parallel Scan
|  |  Index Scan:  Name = DB2INST1.O_OD  ID = 1
|  |  |  Index Columns:
|  |  |  |  1: O_ORDERDATE (Ascending)
|  |  |  #Key Columns = 1
|  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  1: 1998-01-01
|  |  |  |  Stop Key: End of Index
|  |  |  Index-Only Access
|  |  |  Index Prefetch: Eligible 88
|  |  |  |  Insert Into Sorted Shared Temp Table  ID = t1
|  |  |  |  |  #Columns = 1
|  |  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  |  Key 1: (Ascending)
|  |  |  |  |  Use Partitioned Sort
|  |  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  |  #Rows     = 133417
|  |  |  |  |  |  Row Width = 12
|  |  |  |  |  Piped
|  |  Isolation Level: Uncommitted Read
|  |  Lock Intents
|  |  |  Table: Intent None
|  |  |  Row  : None
|  Sorted Shared Temp Table Completion  ID = t1
|  List Prefetch RID Preparation
|  Insert Into Sorted Temp Table  ID = t2
|  |  #Columns = 2
|  |  #Sort Key Columns = 1
|  |  |  Key 1: O_TOTALPRICE (Ascending)
|  |  Sortheap Allocation Parameters:
```

```
|  |  |  #Rows     = 133417
|  |  |  Row Width = 16
|  |  Piped
|  Access Temp Table  ID = t2
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Nested Loop Join
|  |  Access Table Name = DB2INST1.LINEITEM  ID = 4,7
|  |  |  #Columns = 1
|  |  |  Index Scan:  Name = DB2INST1.L_EP  ID = 1
|  |  |  |  Index Columns:
|  |  |  |  |  1: L_EXTENDEDPRICE (Ascending)
|  |  |  |  #Key Columns = 1
|  |  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  |  Stop Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  Index-Only Access
|  |  |  |  Index Prefetch: Eligible 5238
|  |  |  |  |  Insert Into Sorted Temp Table  ID = t3
|  |  |  |  |  |  #Columns = 2
|  |  |  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  |  |  Key 1: (Ascending)
|  |  |  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  |  |  #Rows     = 104448
|  |  |  |  |  |  |  Row Width = 12
|  |  |  |  |  |  Piped
|  |  |  |  |  |  Buffered Partial Aggregation
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  Sorted Temp Table Completion  ID = t3
|  Access Temp Table  ID = t3
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 2
|  Output Sorted
|  |  #Key Columns = 1
|  |  |  Key 1: (Ascending)
Final Aggregation
|  Group By
|  Column Function(s)
Return Data to Application
|  #Columns = 1
```

```
End of section
```

Here is the db2batch output:

```
Elapsed Time is:            14.810      seconds
:
:
Sort heap allocated                          = 0
Total sorts                                  = 6
Total sort time (ms)                         = 3924
Sort overflows                               = 0
:
:
Bufferpool Name                               = IBMDEFAULTBP
:
Buffer pool data logical reads                = 232
Buffer pool data physical reads               = 25
Buffer pool data writes                       = 2
Buffer pool index logical reads               = 50
Buffer pool index physical reads              = 7
Buffer pool index writes                      = 1
Total buffer pool read time (ms)              = 2
Total buffer pool write time (ms)             = 58
:
:
Bufferpool Name                               = TPCDATABP
:
Buffer pool data logical reads                = 22509
Buffer pool data physical reads               = 21793
Buffer pool data writes                       = 0
Buffer pool index logical reads               = 401824
Buffer pool index physical reads              = 5350
Buffer pool index writes                      = 0
Total buffer pool read time (ms)              = 33884
Total buffer pool write time (ms)             = 0
```

In the Explain output, you should notice:

- The estimated cost was slightly reduced (from 55598 to 52211).
- The index on the O_ORDERDATE column was used to access the ORDERS table.
- Before retrieving rows from the ORDERS table, only the row identifiers (RIDs) of the rows which conform the predicate were retrieved from the index and sorted, and then the rows were retrieved using the sorted RIDs.

In the db2batch output, you can see that the performance was not improved very much (from 15.232 sec. to 14.810 sec.).

Here you should consider why the RIDs were taken from the index and sorted before the actual table access is performed. This is probably because of the low cluster ratio of the index. When the cluster ratio is low, the index keys are not in sequential order, but are distributed throughout the table. Therefore, accessing the table in the order of such an index's keys requires more disk I/Os than accessing the table using a high cluster ratio index. Instead of accessing the ORDERS table in the order of the index keys, DB2 decided to get and sort the RIDs before retrieving rows so that the disk I/Os could be minimized.

To increase the cluster ratio, you can reorganize the table using a REORG TABLE command.

9.5.6 Reorganize table

To check the cluster ratio, we executed the following REORGCHK command:

```
REORGCHK CURRENT STATISTICS ON TABLE db2inst1.orders
```

Here is the output.

```
Table statistics:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * TSIZE / ((FPAGES-1) * (TABLEPAGESIZE-76)) > 70
F3: 100 * NPAGES / FPAGES > 80

CREATOR   NAME                  CARD    OV    NP    FP    TSIZE  F1  F2 F3 REORG
--------------------------------------------------------------------------------
DB2INST1  ORDERS             1500000     0 21835 21836 1.76e+08   0  99  99 ---
--------------------------------------------------------------------------------

Index statistics:

F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
F5: 100 * (KEYS * (ISIZE+8) + (CARD-KEYS) * 4) / (NLEAF * INDEXPAGESIZE) > 50
F6: (100-PCTFREE) * (INDEXPAGESIZE-96) / (ISIZE+12) ** (NLEVELS-2) *
(INDEXPAGESIZE-96) / (KEYS * (ISIZE+8) + (CARD-KEYS) * 4) < 100

CREATOR  NAME                 CARD  LEAF  LVLS ISIZE   KEYS   F4   F5  F6 REORG
--------------------------------------------------------------------------------
Table: DB2INST1.ORDERS
DB2INST1 O_OD               2e+06   992     3     4   2406    2   74   61 *--
--------------------------------------------------------------------------------

CLUSTERRATIO or normalized CLUSTERFACTOR (F4) will indicate REORG is necessary
for indexes that are not in the same sequence as the base table. When multiple
indexes are defined on a table, one or more indexes may be flagged as needing
REORG.  Specify the most important index for REORG sequencing.
```

You can see that the cluster ratio (F4) of O_OD is very low (2%). To increase the cluster ratio, we reorganized the ORDERS table using the following command:

```
REORG TABLE db2inst1.orders INDEX db2inst1.o_od
```

You should not forget to execute a RUNSTATS command after reorganizing a table.

```
RUNSTATS ON TABLE db2inst1.orders AND DETAILED INDEXES ALL
```

The access path was changed as in the following graph:

```
                      RETURN
                      (   1)
                        |
                      GRPBY
                      (   2)
                        |
                       LMTQ
                      (   3)
                        |
                      TBSCAN
                      (   4)
                        |
                       SORT
                      (   5)
                        |
                      NLJOIN
                      (   6)
                     /      \
               TBSCAN        IXSCAN
               (   7)        (   6)
                 |          /      \
                SORT   Index:    Table:
               (   8)  DB2INST1  DB2INST1
                 |     L_EP      LINEITEM
               FETCH
               (   9)
              /      \
        IXSCAN        Table:
        (   9)        DB2INST1
       /      \       ORDERS
 Index:    Table:
 DB2INST1  DB2INST1
 O_OD      ORDERS
```

Here is the section information from the Explain output:

```
Intra-Partition Parallelism Degree = 4

Estimated Cost        = 19531
Estimated Cardinality = 104448
```

```
Process Using 4 Subagents
|  Access Table Name = DB2INST1.ORDERS  ID = 4,9
|  |  #Columns = 3
|  |  Parallel Scan
|  |  Index Scan:  Name = DB2INST1.O_OD  ID = 1
|  |  |  Index Columns:
|  |  |  |  1: O_ORDERDATE (Ascending)
|  |  |  #Key Columns = 1
|  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  1: 1998-01-01
|  |  |  |  Stop Key: End of Index
|  |  |  Data Prefetch: Eligible 1942
|  |  |  Index Prefetch: Eligible 1942
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Temp Table  ID = t1
|  |  |  #Columns = 2
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: O_TOTALPRICE (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 133417
|  |  |  |  Row Width = 16
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t1
|  Access Temp Table  ID = t1
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Nested Loop Join
|  |  Access Table Name = DB2INST1.LINEITEM  ID = 4,7
|  |  |  #Columns = 1
|  |  |  Index Scan:  Name = DB2INST1.L_EP  ID = 1
|  |  |  |  Index Columns:
|  |  |  |  |  1: L_EXTENDEDPRICE (Ascending)
|  |  |  |  #Key Columns = 1
|  |  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  |  Stop Key: Inclusive Value
|  |  |  |  |  |  1: ?
|  |  |  |  Index-Only Access
|  |  |  |  Index Prefetch: Eligible 5238
|  |  |  |  |  Insert Into Sorted Temp Table  ID = t2
|  |  |  |  |  |  #Columns = 2
|  |  |  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  |  |  Key 1: (Ascending)
|  |  |  |  |  |  Sortheap Allocation Parameters:
```

```
|  |  |  |  |  |  |  #Rows     = 104448
|  |  |  |  |  |  |  Row Width = 12
|  |  |  |  |  |  Piped
|  |  |  |  |  |  Buffered Partial Aggregation
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  Sorted Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 2
|  Output Sorted
|  |  #Key Columns = 1
|  |  |  Key 1: (Ascending)
Final Aggregation
|  Group By
|  Column Function(s)
Return Data to Application
|  #Columns = 1

End of section
```

In the Explain output, you should notice that:

- The estimated cost was reduced (from 52211 to 19531).
- The RID sort for the orders table no longer exists.

Here is the output from the db2batch:

```
Elapsed Time is:            6.434       seconds
:
:
Sort heap allocated                          = 0
Total sorts                                  = 4
Total sort time (ms)                         = 2111
Sort overflows                               = 0
Active sorts                                 = 0
:
:
Bufferpool Name                               = IBMDEFAULTBP
:
Buffer pool data logical reads                = 214
Buffer pool data physical reads               = 24
Buffer pool data writes                       = 3
Buffer pool index logical reads              = 47
Buffer pool index physical reads             = 8
Buffer pool index writes                     = 0
Total buffer pool read time (ms)             = 2
Total buffer pool write time (ms)            = 68
:
:
Bufferpool Name                              = TPCDATABP
:
Buffer pool data logical reads               = 2317
Buffer pool data physical reads              = 1953
Buffer pool data writes                      = 0
Buffer pool index logical reads              = 401336
Buffer pool index physical reads             = 5350
Buffer pool index writes                     = 0
Total buffer pool read time (ms)             = 5259
Total buffer pool write time (ms)            = 0
```

As the db2batch output shows, the performance goal we had set was finally achieved and we decided to stop tuning the database.

Although we stopped the performance tuning, we could perform further tuning. For example, if we had created an new index whose index keys are O_ORDERDATE, O_TOTALPRICE, and O_CUSTKEY, then the necessary data from the ORDERS table would have been able to be taken from the index only. This could improve the performance further.

Although adding this index could improve the performance, we should mention the overhead of adding the index. By creating indexes, whenever new rows are inserted into the base tables, the new index entries will need to be added. In general, this is something to be careful about. In our case study, this extra overhead for the indexes on the L_EXTENDEDPRICE columns and the one on the O_ORDERDATE columns could be a reasonable trade-off for the improvement in query performance. Creating an index on the O_ORDERDATE, O_TOTALPRICE, and O_CUSTKEY columns could also be an option. However, in real-life situations, the overhead which can be caused by these indexes may be bigger than is justified by the performance improvement, depending on your environment.

Appendix A. Sample scripts

We have written some useful shell scripts to maintain monitored data. These scripts call the Operating System commands or DB2 UDB Monitoring Tools, format the outputs, and save them into the directory structure that we discussed in Chapter 4, "Database design" on page 85.

You can use these sample shell scripts for your environment, or modify them for your own purpose. The source of each sample script can be found on the companion diskette.

A.1 Executing db2look

The `db2look.ksh` script can be used to execute `db2look` with default parameters and place the output in a `states` subdirectory under your home directory. A sample output is included in Chapter 4, "Database design" on page 85.

Here is the syntax:

```
db2look.ksh -c "comment" -d dbname [-p "params"] [-o N] [-v] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -p:   db2look parameters (default="-m -l -a -x -e -f")
        -o:   Save output in dir N under $RESULTS
                (0=Current dir,default; -1=Not Valid)
        -v:   Verbose (default is NOT to display db2look script)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of output filename.
```

Here is the script:

```
#!/bin/ksh
# db2look.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-p \"params\"] [-o N] [-v] [-b]
Execute db2look tool to extract database definitions into an executable script
which can be used to restore the definitions.
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name (required)
    -p:   db2look parameters (default=\"${DB2LOOK_PARAMS}\")
    -o:   Save output in dir N under \$RESULTS
        (0=Current dir,default; -1=Not Valid)
    -v:   Verbose (default is NOT to display db2look script)
    -b:   DONT save prior (old) results to bak directory (default=save)
        (Not saved anyway unless -o is greater than 0)
    Notes:
    Value for -d is part of output filename.
"
}

# Constants

RESULTS=~/states
RESULTS_FILE=`basename $0 .ksh`
RES_EXT=".sql"

# Defaults

QUIET=1
DB_NAME=""
RESULTS_DIR=0# 0 defaults to current dir
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

DB2LOOK_PARAMS="-m -l -a -x -e -f"

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-v") shift; QUIET=0;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *   ) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Database name is required"
```

```
[ $RESULTS_DIR -ge 0 ] 2>/dev/null || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

if [ "$PARSE_ERROR" != "" ]
then
   echo ""
   echo $PARSE_ERROR
   echo ""
   display_syntax
   exit
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`
RES_EXT="_${DB_NAME}${RES_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
   RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
   RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
else
   RES_OUT=${RESULTS_FILE}${RES_EXT}
fi

if [ $RESULTS_DIR -gt 0 ]
then
   mkdir $RESULTS 2>/dev/null
   mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
   if [ $SAVE_OLD_RESULTS -eq 1 ]
   then
      mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
      cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
   fi
fi

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $RES_OUT" || Q_OUTPUT="| tee -a $RES_OUT"
rm $RES_OUT 2>/dev/null

echo "[Creating: $RES_OUT]"

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--             `date`"$Q_OUTPUT
eval echo "-- "      $Q_OUTPUT

eval echo "-- db2look -d $DB_NAME $DB2LOOK_PARAMS"$Q_OUTPUT
eval echo "-- "      $Q_OUTPUT
eval db2look -d $DB_NAME $DB2LOOK_PARAMS$Q_OUTPUT
```

A.2 Executing GET DBM CFG / GET DB CFG

This script executes a `GET DBM CFG` or `GET DB CFG` command and saves the output, at the same time generating a script file which can return all of the parameters to their current value. The generated script can be executed by using the `db2 -tvf` command. In this way, if you experiment with a number of different tunable parameters (not recommended in general, anyway), then you can always return to the values (state) which were saved in this script.

Here is the syntax:

```
SYNTAX: upd_cfg.ksh -c "comment" [-d dbname] [-o N] [-v] [-r] [-b]
Create an SQL script which can be used to restore the current settings of the
DBM CFG or DB CFG.
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name, indicates to use DB CFG (default is DBM CFG)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir,default; -1=Not Valid)
        -v:   Verbose (default is NOT to display generated SQL script)
        -r:   Don't get DB/M CFG, reuse existing output file
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Output of DB/M CFG also saved in -o directory.
        Value for -d is part of output filename and generated script filename.
```

Here is the script:

```
#!/bin/ksh
# upd_cfg.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" [-d dbname] [-o N] [-v] [-r] [-b]
Create an SQL script which can be used to restore the current settings of the
DBM CFG or DB CFG.
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name, indicates to use DB CFG (default is DBM CFG)
    -o:   Save output in dir N under \$RESULTS
        (0=Current dir,default; -1=Not Valid)
    -v:   Verbose (default is NOT to display generated SQL script)
    -r:   Don't get DB/M CFG, reuse existing output file
    -b:   DONT save prior (old) results to bak directory (default=save)
        (Not saved anyway unless -o is greater than 0)
    Notes:
    Output of DB/M CFG also saved in -o directory.
    Value for -d is part of output filename and generated script filename.
"
}

# Constants
```

```
RESULTS=~/states
RESULTS_FILE="dbm_cfg"
RES_EXT=".out"
SUM_EXT=".sql"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=1
DB_NAME=""
RESULTS_DIR=0# 0 defaults to current dir
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

# Parse parameters

while [ "$1" != "" ]
do
   case "$1" in
   "-c") shift; COMMENT=$1; shift;;
   "-d") shift; DB_NAME=$1; shift;;
   "-o") shift; RESULTS_DIR=$1; shift;;
   "-v") shift; QUIET=0;;
   "-r") shift; REUSE_OUT=1;;
   "-b") shift; SAVE_OLD_RESULTS=0;;
   *   ) shift; PARSE_ERROR="Invalid Param";;
   esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
   PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ $RESULTS_DIR -ge 0 ] 2>/dev/null || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

if [ "$PARSE_ERROR" != "" ]
then
   echo ""
   echo $PARSE_ERROR
   echo ""
   display_syntax
   exit
fi

if [ "$DB_NAME" != "" ]
then
   DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`
   RESULTS_FILE="db_cfg"
   RES_EXT="_${DB_NAME}${RES_EXT}"
   SUM_EXT="_${DB_NAME}${SUM_EXT}"
fi

if [ $RESULTS_DIR -gt 0 ]
then
   RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
   SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
   RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
   SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
```

```
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
        mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
        [ $REUSE_OUT -eq 0 ] && \
        cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
        cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
fi

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

echo "[Creating: $SUM_OUT]"

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- --------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--             `date`"$Q_OUTPUT
eval echo "-- "       $Q_OUTPUT

if [ $REUSE_OUT -eq 0 ]
then
    echo "[Creating: $RES_OUT]"
    echo ""
    if [ "$DB_NAME" = "" ]
    then
        eval echo "-- db2 get dbm cfg"$Q_OUTPUT
        db2 get dbm cfg > $RES_OUT
    else
        eval echo "-- db2 get db cfg for $DB_NAME" $Q_OUTPUT
        db2 get db cfg for $DB_NAME> $RES_OUT
    fi
else
    echo "[Reusing: $RES_OUT]"
fi

eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
```

The `upd_cfg.ksh` script uses `upd_cfg.awk` file whose source is as follows:

```
# run: db2 get dbm cfg | awk -f upd_cfg.awk > upd_dbm.sql
# run: db2 get db cfg for xx | awk -f upd_cfg.awk > upd_db_xx.sql

BEGIN{once = 0 ; set_NEWLOGPATH=0 }

/Database Manager Configuration/ && (once == 0){
    print "UPDATE DBM CFG USING";
    once = 1;
    FS=" = ";
}

/Database Configuration for Database/ && (once == 0){
    print "UPDATE DB CFG FOR " $NF " USING";
    once = 1;
    FS=" = ";
}

{   # Print the original line as a comment
    print "--" $0;
}

# Look for configurable parameters in parenthases
/\([A-Z]+[A-Z_0-9]*\)/{
    match( $1, /\([A-Z]+[A-Z_0-9]*\)/ );

    # pull out parameter name and current value
    parm = substr( $1, RSTART+1, RLENGTH-2);
    val  = $2;

    # If the value is blank, set to empty string
    if (val ~ /^ *$/) {
        # Exception made for NEWLOGPATH, since empty string forces
        # value of a default path
        if (parm == "NEWLOGPATH") {
            set_NEWLOGPATH=1;
            next; # Skip the rest, don't print anything, see below
        } else {
            val = "''";
        }
    } else {
        # Remove part of value btwn paren's. Force recalc (-1) when:
        # 1) value contains "(caluculated)"; 2) value starts with "MAX";
        # 3) entire value was between paren's
        gsub( /\(.*\)/, "", $2);
        if ( (val ~ /\(calculated\)/) || (val ~ /^MAX/) ||
             ($2 ~ /^ *$/) ) {
            val = "-1";
        } else {
            val = $2;
        }
    }

    # Print the uncommented line
    printf "%57.57s    %s\n", parm, val;
}

# Exception for NEWLOGPATH: Set it to the current log path
/Path to log files/ && set_NEWLOGPATH{
    printf "%57.57s    %s\n", "NEWLOGPATH", $2;
}

END{ print ";" }
```

A.3 Display statements in the dynamic SQL cache

The sample script sqlcache.ksh displays the SQL statements and selected statistics currently in the dynamic SQL cache.

Here is the syntax:

```
sqlcache.ksh -c "comment" -d dbname [-o N] [-f] [-s N] [-sql]
              [-t] [-w] [-q] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir; -1=Not Saved,default)
        -f:   Save each SQL stmt to a different file in the $QUERIES dir
        -s:   Display N characters of SQL stmt (-1=all, default)
        -sql: Only display SQL statements, without statistics
        -t:   DONT display Timing stats
        -w:   DONT display Row Count stats
        -q:   Quiet (default is to display output)
        -r:   Don't get snapshot, reuse existing snapshot output file
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of snapshot output filename.
        Values for -d, -s, -f and -sql are part of Summary output filename.
        Timing and Row Count statistics won't be shown if "Not Collected".
        In most cases, best viewed in 80 or 132 column window.
```

The source is as follows:

```
#!/bin/ksh
# sqlcache.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
   echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-o N] [-f] [-s N] [-sql]
        [-t] [-w] [-q] [-r] [-b]
Summerize the Statement statistics captured by the Dynamic SQL Snapshot.
   -c:   Comment placed on first line of Summary output file (required)
   -d:   Database name (required)
   -o:   Save output in dir N under \$RESULTS
      (0=Current dir; -1=Not Saved,default)
   -f:   Save each SQL stmt to a different file in the \$QUERIES dir
   -s:   Display N characters of SQL stmt (-1=all, default)
   -sql: Only display SQL statements, without statistics
   -t:   DONT display Timing stats
   -w:   DONT display Row Count stats
   -q:   Quiet (default is to display output)
   -r:   Don't get snapshot, reuse existing snapshot output file
   -b:   DONT save prior (old) results to bak directory (default=save)
      (Not saved anyway unless -o is greater than 0)
   Notes:
   Value for -d is part of snapshot output filename.
   Values for -d, -s, -f and -sql are part of Summary output filename.
```

```
    Timing and Row Count statistics won't be shown if \"Not Collected\".
    In most cases, best viewed in 80 or 132 column window.
"
}

# Constants

QUERIES=~/queries
RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
RES_EXT=".out"
SUM_EXT=".sum"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=0
DISPLAY_SQL=-1
SAVE_SQL=0
SQL_ONLY=0
RESULTS_DIR=-1# -1 defaults to not saved
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

NO_TIMING=0;
NO_ROWS=0;
WINCOLS=`stty size | awk '{print $2}'`

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-f") shift; SAVE_SQL=1;;
    "-s") shift; DISPLAY_SQL=$1; shift;;
    "-sql") shift; SQL_ONLY=1;;
    "-t") shift; NO_TIMING=1;;
    "-w") shift; NO_ROWS=1;;
    "-q") shift; QUIET=1;;
    "-r") shift; REUSE_OUT=1;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *   ) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ $DISPLAY_SQL -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -s param"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"
```

```
echo "$PARAMS" | awk '/-o -1/ && /-r/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -r with -o -1"

echo "$PARAMS" | awk '/-o -1/ && /-q/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -o -1"

[ $SQL_ONLY -eq 1 ] && NO_TIMING=0 && NO_TIMING=0

echo "$PARAMS" | awk '/-sql/ && (/-t/ || /-w/){exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -sql with -t or -w"

if [ "$PARSE_ERROR" != "" ]
then
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

if [ $SAVE_SQL -eq 1 ]
then
    # Get last query file number
    mkdir $QUERIES 2>/dev/null
    LAST_QUERY=`ls $QUERIES | sort -n | tail -1`
    LAST_QUERY=`basename $LAST_QUERY .sql`
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}_${DISPLAY_SQL}${SAVE_SQL}${SQL_ONLY}${NO_TIMING}${NO_ROWS}${SUM_EXT
}"

if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
        mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
        [ $REUSE_OUT -eq 0 ] && \
        cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
        cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
```

```
fi

TMP_SQL="${QUERIES}/sql.$$.tmp"

# Clean up previous aborts (trap didn't work :(
rm ${QUERIES}/sql.[0-9]*.tmp 2>/dev/null

export QUERIES DISPLAY_SQL SAVE_SQL SQL_ONLY LAST_QUERY TMP_SQL
export NO_TIMING NO_ROWS WINCOLS

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

if [ $RESULTS_DIR -ge 0 ]
then
   echo "[Creating: $SUM_OUT]"
else
   Q_OUTPUT=""
   echo "[No Output Saved]"
fi

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--             `date`"$Q_OUTPUT
eval echo "-- "       $Q_OUTPUT

if [ $RESULTS_DIR -eq -1 ]
then
   echo db2 get snapshot for dynamic sql on $DB_NAME
   echo ""
   db2 get snapshot for dynamic sql on $DB_NAME | awk -f $AWKSCRIPT
else
   if [ $REUSE_OUT -eq 0 ]
   then
      eval echo db2 get snapshot for dynamic sql on $DB_NAME $Q_OUTPUT
      eval echo ""        $Q_OUTPUT
      echo "[Creating: $RES_OUT]"
      db2 get snapshot for dynamic sql on $DB_NAME > $RES_OUT
   else
      echo ""
      echo "[Reusing: $RES_OUT]"
   fi

   eval echo ""          $Q_OUTPUT
   eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
fi
```

The `sqlcache.ksh` file uses the following `sqlcache.awk` file:

```
BEGIN{
   OS      = "AIX";
   QUERIES     = ENVIRON[ "QUERIES"     ];
   DISPLAY_SQL = ENVIRON[ "DISPLAY_SQL" ];
   SAVE_SQL    = ENVIRON[ "SAVE_SQL"    ];
   SQL_ONLY    = ENVIRON[ "SQL_ONLY"    ];
   NO_TIMING   = ENVIRON[ "NO_TIMING"   ];
   NO_ROWS     = ENVIRON[ "NO_ROWS"     ];
   LAST_QUERY  = ENVIRON[ "LAST_QUERY"  ];
   TMP_SQL     = ENVIRON[ "TMP_SQL"     ];
   WINCOLS     = ENVIRON[ "WINCOLS"     ];
```

```
    header    = 0;
    COLLECTED   = 1;
    STMT_NUM    = 0;
    MIL         = 1000000;

#print  QUERIES    "=QUERIES"     ;
#print  DISPLAY_SQL "=DISPLAY_SQL" ;
#print  SAVE_SQL    "=SAVE_SQL"    ;
#print  SQL_ONLY    "=SQL_ONLY"    ;
#print  NO_TIMING   "=NO_TIMING"   ;
#print  NO_ROWS     "=NO_ROWS"     ;
#print  LAST_QUERY  "=LAST_QUERY"  ;
#print  TMP_SQL     "=TMP_SQL"     ;
#print  WINCOLS     "=WINCOLS"     ;
}

/SQL1611W/{ print }

/ Number of executions/{
                    N_EXECUTIONS= "";
                    N_COMPILES= "";
                    T_W_PREP= "";
                    T_B_PREP= "";
                    R_DELETED= "";
                    R_INSERTED= "";
                    R_READ= "";
                    R_UPDATED= "";
                    R_WRITTEN= "";
                    S_SORTS= "";
                    T_T_EXECUTION= "";
                    T_T_USER= "";
                    T_T_SYSTEM= "";
}
/ Number of executions/{ N_EXECUTIONS=$NF}
/ Number of compilations/{ N_COMPILES=$NF}
/ Worst preparation time \(ms\)/{ T_W_PREP=$NF}
/ Best preparation time \(ms\)/{ T_B_PREP=$NF}
/ Rows deleted/{ R_DELETED=$NF}
/ Rows inserted/{ R_INSERTED=$NF}
/ Rows read/  { R_READ=$NF}
/ Rows updated/{ R_UPDATED=$NF}
/ Rows written/{ R_WRITTEN=$NF}
/ Statement sorts/{ S_SORTS=$NF}
/ Total execution time \(sec.ms\)/{ T_T_EXECUTION=$NF}
/ Total user cpu time \(sec.ms\)/{ T_T_USER=$NF}
/ Total system cpu time \(sec.ms\)/{ T_T_SYSTEM=$NF}

/ Statement text/{  # Begin Display

    STMT_NUM=STMT_NUM + 1;
    S_TEXT=substr( $0, index( $0, "=")+2 );
    SQL_OUT="";

    if (R_READ == "Collected")
        COLLECTED=0;

    if (DISPLAY_SQL == -1)
        SQL_LEN = length( S_TEXT);
    else
        SQL_LEN = DISPLAY_SQL;

    # Save SQL Text to a file, or get name of existing (duplicate) file
    if ( SAVE_SQL )  Save_Sql();
    headline = "";
    dataline = "";

    if ( SQL_ONLY )
    {
        if (SAVE_SQL)S_Save_Info();
    }
    else
    {
        T_A_EXEC= T_T_EXECUTION / N_EXECUTIONS;
        T_A_USER= T_T_USER/ N_EXECUTIONS;
        T_A_SYSTEM= T_T_SYSTEM / N_EXECUTIONS;
        A_R_READ= R_READ/ N_EXECUTIONS;
        A_R_WRITTEN= R_WRITTEN/ N_EXECUTIONS;
        A_R_INSERTED= R_INSERTED/ N_EXECUTIONS;
        A_R_UPDATED= R_UPDATED/ N_EXECUTIONS;
        A_R_DELETED= R_DELETED/ N_EXECUTIONS;

        N_EXECUTIONS= round_MIL( N_EXECUTIONS );
        N_COMPILES= round_MIL( N_COMPILES   );
        A_R_READ= round_MIL( A_R_READ     );
        A_R_WRITTEN= round_MIL( A_R_WRITTEN  );
        A_R_INSERTED= round_MIL( A_R_INSERTED );
        A_R_UPDATED= round_MIL( A_R_UPDATED  );
        A_R_DELETED= round_MIL( A_R_DELETED  );

        S_Exec_Info();

        if (COLLECTED && ! NO_TIMING) S_Timing_Info();

#       if ( SAVE_SQL ||
#            ((! SAVE_SQL) && ! (NO_TIMING || NO_ROWS || ! COLLECTED)) )
        if ( SAVE_SQL )
            S_Save_Info();

        if (COLLECTED && ! NO_ROWS) S_Rows_Info();
    }

    S_SQL_Text();

    rm_Trailing();

    if ( headline && ! header)
    {
        for (i=1; i<=length( headline); i++)
            underline = underline "-";

        print headline;
        print underline;
        header = 1;
    }

    if (dataline) print dataline;
}

################################################################################
function round_MIL( val ) {

    if ( val > MIL ) val = int( val / MIL ) "M";
    return val;
}
```

```
################################################################################
function S_Exec_Info() {

   headline = headline sprintf( \
      "%4.4s | %6.6s %6.6s | %10.10s | ",
      "Qnum",
      "Exec's",
      "Comp's",
      "BestPrepMS");
   dataline = dataline sprintf( \
      "|%3s | %6s %6s | %10s | ",
      STMT_NUM,
      N_EXECUTIONS,
      N_COMPILES,
      T_B_PREP);
}

################################################################################
function S_Timing_Info() {

   headline = headline sprintf( \
      "%10.10s %10.10s %10.10s | ",
      "ExecSEC.MS",
      "UserSEC.MS",
      "SystSEC.MS");
   dataline = dataline sprintf( \
      "%10.10s %10.10s %10.10s | ",
      T_A_EXEC,
      T_A_USER,
      T_A_SYSTEM);
}

################################################################################
function S_Rows_Info() {

   SPACERHEAD="";
   SPACER="";
   WINCOL_Spacing();

   headline = headline sprintf( \
      "%6.6s %6.6s %s%10.10s   %10.10s %10.10s | ",
      "R_Read",
      "Writtn",
      SPACERHEAD,
      "Inserted",
      "Updated",
      "Deleted");
   dataline = dataline sprintf( \
      "%6.6s %6.6s %s%10.10s   %10.10s %10.10s | ",
      A_R_READ,
      A_R_WRITTEN,
      SPACER,
      A_R_INSERTED,
      A_R_UPDATED,
      A_R_DELETED);
}

################################################################################
function S_Save_Info() {

   headline = headline sprintf( \
         "%8.8s | ",
         "SQL-File");
```

```
#   if ( ( ! SQL_ONLY ) || ( SQL_ONLY && S_TEXT ) )
    dataline = dataline sprintf( \
            "%8s | ",
            SQL_OUT);
}

################################################################################
function S_SQL_Text() {

    if ( DISPLAY_SQL )
        headline = headline sprintf( \
            "%-*s | ",
            SQL_LEN, "SQL-Text");
    if (S_TEXT && SQL_LEN)
        dataline = dataline sprintf( \
            "%-*.*s | ",
            SQL_LEN, SQL_LEN, S_TEXT);
}

################################################################################
function WINCOL_Spacing() {

    if ( COLLECTED && ! ( NO_TIMING || NO_ROWS ) )
    {
        if (SAVE_SQL)
        {
            if (WINCOLS == 80)
            {
                headline = headline "    | ";
                dataline = dataline "    | ";
                SPACERHEAD="| ";
                SPACER="| ";
            }
            else if (WINCOLS == 132)
            {
                SPACERHEAD="|";
                SPACER="|";
            }
        }
        else # No SAVE_SQL info
        {
            if (WINCOLS == 80)
            {
                headline = headline "          |     | ";
                dataline = dataline "          |     | ";
                SPACERHEAD="| ";
                SPACER="| ";
            }
            else if (WINCOLS == 132)
            {
                headline = headline "         | ";
                dataline = dataline "         | ";
                SPACERHEAD="| ";
                SPACER="| ";
            }
        }
    }
}

################################################################################
function rm_Trailing() {
# Get rid of trailing separator
```

```
    if ( substr( headline, (len=length( headline))-2) == " | " )
    {
        headline = substr( headline, 1, (len - 3) );
        dataline = substr( dataline, 1, (len - 3) );
    }
}

#########################################################################
function Save_Sql() {
# Save SQL Text to a file, or get name of existing (duplicate) file

    DUPFILE="";
    print S_TEXT > TMP_SQL;
    close( TMP_SQL);

    # Check if a duplicate SQL file exists
    if ( OS == "AIX" )
    {
        # Get filesize of TMP_SQL
        "ls -l " TMP_SQL | getline DIRLINE;
        close( "ls -l " TMP_SQL );
        DFN=split( DIRLINE, DIRARRAY);
        FILESIZE=DIRARRAY[5];

        # Loop thru sql files in reverse creation order
        # Compare only if same filesize
        while ( (DUPFILE == "") &&
            ("ls -lt " QUERIES "/[0-9]*.sql" | getline DIRLINE ) )
        {
            DFN=split( DIRLINE, DIRARRAY);
            if ((DIRARRAY[5] == FILESIZE) &&
                 (TMP_SQL != DIRARRAY[DFN])  )
            {
                "diff " TMP_SQL " " DIRARRAY[DFN]   \
                " >/dev/null 2>&1 && "              \
                "  basename "  DIRARRAY[DFN] | \
                getline DUPFILE;
            close(  "diff " TMP_SQL " " DIRARRAY[DFN]   \
                " >/dev/null 2>&1 && "              \
                "  basename "  DIRARRAY[DFN] );
            }
        }
        close( "ls -lt " QUERIES "/[0-9]*.sql" );

        system( "rm " TMP_SQL);
    }
    else# Untested
    {
        # delete TMP_SQL
    }

    if ( DUPFILE != "" )
    {  # A duplicate SQL file was found
        SQL_OUT = DUPFILE;
    }
    else
    {  # Create SQL file
        LAST_QUERY=LAST_QUERY + 1;
        SQL_OUT=LAST_QUERY ".sql";
        print S_TEXT > QUERIES "/" SQL_OUT;
    }
}
```

A.4 Disk I/O activity

The script `iostat.ksh` traps `iostat` output and displays only part of the information horizontally, making it easier to track changes in activity. The output of the `iostat.ksh` script is saved under the `results` directory.

Here is the syntax:

```
iostat.ksh -c "comment" [-i m] [-p] [-o N] [-s n ] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required).
        -i:   Interval for iostat (default=1).
        -p:   Only display peak values, when they change, for all disks, etc.
              Default is to display all values, as they change.
        -o:   Save output in dir N under $RESULTS (-1=current dir, default).
        -s:   Save iostat output with n in the filename (0=dont save,default).
        -r:   Don't get iostat, reuse iostat output specifies by -s.
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Stop the utility by hitting Ctrl-C.
        Values for -i, -s and -p are part of Summary output filename.
        In most cases, best viewed in window-width which is multiple of 10.
```

The source is as follows:

```
#!/bin/ksh
# iostat.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
   echo "\
SYNTAX: `basename $0` -c \"comment\" [-i m] [-p] [-o N] [-s n ] [-r] [-b]
Summerize the output of iostat.
   -c:   Comment placed on first line of Summary output file (required).
   -i:   Interval for iostat (default=1).
   -p:   Only display peak values, when they change, for all disks, etc.
             Default is to display all values, as they change.
   -o:   Save output in dir N under \$RESULTS (-1=current dir, default).
   -s:   Save iostat output with n in the filename (0=dont save,default).
   -r:   Don't get iostat, reuse iostat output specifies by -s.
   -b:   DONT save prior (old) results to bak directory (default=save)
      (Not saved anyway unless -o is greater than 0)
   Notes:
   Stop the utility by hitting Ctrl-C.
   Values for -i, -s and -p are part of Summary output filename.
   In most cases, best viewed in window-width which is multiple of 10.
"
}

# Constants

RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
RES_EXT=".out"
SUM_EXT=".sum"
```

```
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

RESULTS_DIR=-1# -1 defaults to current dir
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

INTERVAL=1
SAVE_NAME=0
ONLYPEAKS=0

# Parse parameters

while [ "$1" != "" ]
do
   case "$1" in
   "-c") shift; COMMENT=$1; shift;;
   "-i") shift; INTERVAL=$1; shift;;
   "-p") shift; ONLYPEAKS=1;;
   "-o") shift; RESULTS_DIR=$1; shift;;
   "-s") shift; SAVE_NAME=$1;   shift;;
   "-r") shift; REUSE_OUT=1; ;;
   "-b") shift; SAVE_OLD_RESULTS=0;;
   *   ) shift; PARSE_ERROR="Invalid Param";;
   esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
   PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

[ $INTERVAL -ge 1 ] 2>/dev/null || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid number following -i param"

[ $SAVE_NAME -ge 0 ] 2>/dev/null || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid number following -s param"

[ $REUSE_OUT -eq 1 -a $SAVE_NAME -eq 0 ] && \
   PARSE_ERROR="${PARSE_ERROR} -Cant use -r with -s 0 or missing -s"

if [ "$PARSE_ERROR" != "" ]
then
   echo ""
   echo $PARSE_ERROR
   echo ""
   display_syntax
   exit
fi

RES_EXT="_${INTERVAL}${SAVE_NAME}${RES_EXT}"
SUM_EXT="_${INTERVAL}${SAVE_NAME}${ONLYPEAKS}${SUM_EXT}"

if [ $RESULTS_DIR -ge 0 ]
then
   RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
   SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
```

```
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$

    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
        mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
        [ $REUSE_OUT -eq 0 ] && \
        cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
        cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

export ONLYPEAKS

# BEGIN .......

[ $SAVE_NAME -eq 0 ] && TEE_OUT="" || TEE_OUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

[ $SAVE_NAME -ne 0 ] && echo "[Creating: $SUM_OUT]"

eval echo "$COMMENT"$TEE_OUT
eval echo "-------------------------------------------"$TEE_OUT
eval echo "Invocation: $0 $PARAMS"$TEE_OUT
eval echo "             `date`"$TEE_OUT
eval echo             $TEE_OUT

if [ $REUSE_OUT -eq 1 ]
then
    [ ! -f $RES_OUT ] && echo "Can't reuse $RES_OUT - Missing" && exit

    awk -f $AWKSCRIPT $RES_OUT | tee -a $SUM_OUT
else
    echo "Hit Ctrl-C to stop..."
    trap "" 1 2 3
    if [ $SAVE_NAME -eq 0 ]
    then
        echo "iostat output and summary not saved"
        echo "Waiting for disk i/o ..."

        iostat $INTERVAL | awk -f $AWKSCRIPT
    else
        echo "[Creating: $RES_OUT]"
        echo "Waiting for disk i/o ..."

        iostat $INTERVAL | tee $RES_OUT | awk -f $AWKSCRIPT
        awk -f $AWKSCRIPT $RES_OUT >> $SUM_OUT
    fi
fi
```

The `iostat.ksh` script uses the following `iostat.awk` file:

```
# iostat_disk.awk
# Raanon Reutlinger, IBM Israel, May 2000

BEGIN{
       HEADCOUNTMAX=10;  # Heading every x lines
       ONLYPEAKS=ENVIRON[ "ONLYPEAKS"];  # All changes or all Peaks
```

```
#       ALLCHANGES=ENVIRON[ "ALLCHANGES"];# All changes or all Highest
        IOWAIT="IOWAIT";  # Heading for IOWAIT
        changedflag=0;
        firstdisk="";
        lastdisk="";
        last_iowait=0;
        cnt=0;
        if (ONLYPEAKS == "1")
            ALLCHANGES=0;
        else
            ALLCHANGES=1;  # Default is ALLCHANGES=YES
}

/^Disks:/ || /^$/{ next }

/^tty:/{

    ++cnt;
    get_iowait=1;
    next;
}

get_iowait{

    get_iowait=0;
    iowait=$NF;
    next;
}

(cnt == 1){ lastdisk=$1 }

# Skip the first two entries of every disk - ramp up of values
(cnt > 2){

    if ( (iowait > 0) && ( ALLCHANGES || (iowait > last_iowait) ) )
    {
        changed[IOWAIT]=cnt;
        reads[IOWAIT]=iowait;
        writes[IOWAIT]=iowait;
        last_iowait=iowait;
        changedflag=1;
    }
    if ( ($5 > 0) && ( ALLCHANGES || ($5 > reads[$1]) ) )
    {
        changed[$1]=cnt;
        reads[$1]=$5;
        changedflag=1;
    }
    if ( ($6 > 0) && ( ALLCHANGES || ($6 > writes[$1]) ) )
    {
        changed[$1]=cnt;
        writes[$1]=$6;
        changedflag=1;
    }
}

($1 == lastdisk) && (changedflag){  display_results( "-" ) }

END{
    if (! ALLCHANGES)
    {
        headcount = HEADCOUNTMAX - 1;
        display_results( "=" );
    }
}

###########################
function display_results( underline_char )
{
    # Build heading, then replace spaces with dashes
    headcount+=1;
    headline1="";
    for (disk in changed)
        if (changed[disk] > 0)
            headline1 = headline1 sprintf( " %-9.9s", disk);
    gsub( / /, underline_char, headline1);
    if ( (headline1 != headline) || (headcount == HEADCOUNTMAX) )
    {
        headline = headline1;
        print headline;
        headcount=0;
    }

    display( "R", reads);
    display( "W", writes);
    changedflag=0;
}

############################
function display( RW, activity )
{
    got_activity=0;
    line="";
    for (disk in changed)
    {
        if (changed[disk] > 0)
        {
            if ( (activity[disk]) && (changed[disk] || ! ALLCHANGES))
            {
            line = line sprintf( "%s%9s", RW, activity[disk]);
            if (disk != IOWAIT)
                got_activity=1;
            }
            else
            line = line sprintf( "%s%9s", RW, "");
            RW="|";
        }
        if (ALLCHANGES)
            activity[disk]="";
    }
    if (got_activity)
        print line;
}
```

A.5 Lock information

The `locks.ksh` script summarizes the current locks information being captured by the locks snapshot monitor. All of the information displayed is collected even without turning the LOCKS monitor switch ON. The number of locks are displayed at the database, application and table levels.

Here is the syntax:

```
locks.ksh -c "comment" -d dbname [-o N] [-a] [-db] [-q] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir; -1=Not Saved,default)
        -a:   DONT display Application level information
        -db:  DONT display Database level information
        -q:   Quiet (default is to display output)
        -r:   Don't get snapshot, reuse existing snapshot output file
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of snapshot output filename.
        Values for -d, -a and -db are part of Summary output filename.
```

The following is the source:

```
#!/bin/ksh
# locks.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
    echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-o N] [-a] [-db] [-q] [-r] [-b]
Summerize the LOCKS information being captures by a LOCKS snapshot.
    -c:   Comment placed on first line of Summary output file (required)
    -d:   Database name (required)
    -o:   Save output in dir N under \$RESULTS
        (0=Current dir; -1=Not Saved,default)
    -a:   DONT display Application level information
    -db:  DONT display Database level information
    -q:   Quiet (default is to display output)
    -r:   Don't get snapshot, reuse existing snapshot output file
    -b:   DONT save prior (old) results to bak directory (default=save)
        (Not saved anyway unless -o is greater than 0)
    Notes:
    Value for -d is part of snapshot output filename.
    Values for -d, -a and -db are part of Summary output filename.
"
}

# Constants

RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
```

```
RES_EXT=".out"
SUM_EXT=".sum"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=0
RESULTS_DIR=-1# -1 defaults to not saved
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

NO_APPS=0;
NO_DB=0;

# Parse parameters

while [ "$1" != "" ]
do
    case "$1" in
    "-c") shift; COMMENT=$1; shift;;
    "-d") shift; DB_NAME=$1; shift;;
    "-o") shift; RESULTS_DIR=$1; shift;;
    "-a") shift; NO_APPS=1;;
    "-db") shift; NO_DB=1;;
    "-q") shift; QUIET=1;;
    "-r") shift; REUSE_OUT=1;;
    "-b") shift; SAVE_OLD_RESULTS=0;;
    *   ) shift; PARSE_ERROR="Invalid Param";;
    esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
    PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
    PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

echo "$PARAMS" | awk '/-o -1/ && /-r/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -r with -o -1"

echo "$PARAMS" | awk '/-o -1/ && /-q/{exit -1}' || \
    PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -o -1"

if [ "$PARSE_ERROR" != "" ]
then
    echo ""
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}_${NO_APPS}${NO_DB}${SUM_EXT}"
```

```
if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
        mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
        [ $REUSE_OUT -eq 0 ] && \
        cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
        cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
fi

export NO_APPS NO_DB

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

if [ $RESULTS_DIR -ge 0 ]
then
    echo "[Creating: $SUM_OUT]"
else
    Q_OUTPUT=""
    echo "[No Output Saved]"
fi

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ----------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--             `date`"$Q_OUTPUT
eval echo "-- "       $Q_OUTPUT

if [ $RESULTS_DIR -eq -1 ]
then
    echo db2 get snapshot for locks on $DB_NAME
    echo ""
    db2 get snapshot for locks on $DB_NAME | awk -f $AWKSCRIPT
else
    if [ $REUSE_OUT -eq 0 ]
    then
        eval echo db2 get snapshot for locks on $DB_NAME $Q_OUTPUT
        echo ""
        echo "[Creating: $RES_OUT]"
        db2 get snapshot for locks on $DB_NAME > $RES_OUT
    else
        echo ""
        echo "[Reusing: $RES_OUT]"
    fi

    eval echo ""          $Q_OUTPUT
    eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
fi
```

The `locks.ksh` script uses the following `locks.awk` file:

```
BEGIN{
    NO_APPS  = ENVIRON[ "NO_APPS"   ];
    NO_DB    = ENVIRON[ "NO_DB"     ];
    headline = "";
    dashes   = "-----------------------------";
    print "";
}

/SQL1611W/{ print }

# Snapshot info at database level
!NO_DB && /Database name /{ print }
!NO_DB && /Database path /{ print }
!NO_DB && /Input database alias /{ print }
!NO_DB && /Locks held / && ! A_HAND{ print }
!NO_DB && /Applications currently connected /{ print }
!NO_DB && /Agents currently waiting on locks /{ print }
!NO_DB && /Snapshot timestamp /{ print ; print "" }

# Application info
/Application handle /{ A_HAND = ""  ;
             A_NAME= ""  ;
             A_USER= ""  ;
             H_HAND= ""  ;
             A_ID = ""  ;
             A_STATUS= ""  ;
             A_LOCKS= ""  ;
             A_WAIT= ""  ;
}
/Application handle /{ A_HAND = $NF }
/Application ID /{ A_ID = $NF ; sub( /\.[0-9]*$/, "", A_ID) }
/Application name /{ A_NAME = $NF }
/Authorization ID /{ A_USER = $NF }
/Application status /{ A_STATUS = substr( $0, index( $0, "=") + 2 )  }
/Status change time /{ A_TIME = $NF }
/Locks held / && A_HAND{ A_LOCKS = $NF }
/Total wait time \(ms\)/{ A_WAIT = $NF }

/Total wait time \(ms\)/ && (! NO_APPS) {

    if (A_WAIT == "Collected")  A_WAIT = "";

    if (! headline )
    {
        headline = sprintf( \
        "%-10.10s %-8.8s %-6.6s %-20.20s %-15.15s %-5.5s %-7.7s",
            "APP.NAME",
            "APP.USER",
            "HANDLE",
            "APP.ID",
            "APP.STATUS",
            "LOCKS",
            "WAIT.ms");

        underline = sprintf( \
        "%-10.10s %-8.8s %-6.6s %-20.20s %-15.15s %-5.5s %-7.7s",
            dashes,
            dashes,
            dashes,
            dashes,
            dashes,
            dashes,
            dashes);

        print headline;
        print underline;
    }
    dataline = sprintf( \
    "%-10.10s %-8.8s %6.6s %-20.20s %-15.15s %5.5s %s",
        A_NAME,
        A_USER,
        A_HAND,
        A_ID,
        A_STATUS,
        A_LOCKS,
        A_WAIT);
    print dataline;
}

# Lock info
/ Object Type /{ L_TYPE = ""  ;
             L_SCHEMA = ""  ;
             L_TABLE = ""  ;
             L_MODE = ""  ;
             L_STATUS = ""  ;
             L_ESC = ""  ;
}
/ Object Type /{ L_TYPE = $NF }
/ Table Schema /{ L_SCHEMA = $NF }
/ Table Name /{ L_TABLE = $NF }
/ Mode /{ L_MODE = $NF }
/ Status /{ L_STATUS = $NF }
/ Lock Escalation /{ L_ESC = $NF ;
             LOCKS [ L_SCHEMA ,
                  L_TABLE  ,
                  L_TYPE   ,
                  L_ESC    ,
                  L_MODE   ,
                  L_STATUS ] ++;
}

END{
    headline = "";
    for (ind in LOCKS)
    {
        if (! headline)
        {
            headline = sprintf( \
        "%-30.30s | %-10.10s | %-3.3s | %-4.4s | %-10.10s | %5.5s",
            "TABLE NAME",
            "TYPE",
            "ESCALATED",
            "MODE",
            "STATUS",
            "COUNT");

            underline = sprintf( \
        "%-30.30s | %-10.10s | %-3.3s | %-4.4s | %-10.10s | %5.5s",
            dashes,
            dashes,
            dashes,
            dashes,
            dashes,
            dashes);

#           underline = headline;
#           gsub( /./, "-", underline);

            print "" ;
            print headline;
            print underline;
        }

        split( ind, LOCK_INFO, SUBSEP);
        SCHEMA_TABLE = LOCK_INFO[1] "." LOCK_INFO[2];
        dataline = sprintf( \
        "%-30.30s | %-10.10s | %-3.3s | %-4.4s | %-10.10s | %5.5s",
            SCHEMA_TABLE,
            LOCK_INFO[3],
            LOCK_INFO[4],
            LOCK_INFO[5],
            LOCK_INFO[6],
            LOCKS[ind]);
        print dataline;
    }

    if (! headline)
    {
        print "";
        print "*** NO LOCKS ***";
    }
}
```

A.6 Statement Event Monitor

The `mon_stmt.ksh` script returns information about the SQL statements (for example, `OPEN CURSOR`, `FETCH`) reaching the DB2 engine as collected by the statements Event Monitor. This sample script issues the `db2evmon` command which reads the event records and generates a report. Before using the script, the Event Monitor has to be created and activated.

Here is the syntax:

```
mon_stmt.ksh -c "comment" -d dbname -m mon_name [-o N] [-f]
              [-s N] [-sql] [-i Q] [-p] [-ps] [-q] [-r] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -m:   Monitor name (required)
        -o:   Save output in dir N under $RESULTS
                (0=Current dir; -1=Not Saved,default)
        -f:   Save each SQL stmt to a different file in the $QUERIES dir
        -s:   Display N characters of SQL stmt (-1=all, default)
        -sql: Only display SQL statements, without statistics
        -i:   Include info in output, where Q could contain any of the following
               S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info
               N=None_of_the_above (default=SORTX)
        -p:   mon_name is a PIPE; Don't save the summary to an output file.
        -ps:  mon_name is a PIPE; Save the summary to an output file.
               Summary output is only visible when the event monitor closes.
        -q:   Quiet (default is to display output)
        -r:   Don't read the monitor files, reuse existing output (extraction)
        -b:   DONT save prior (old) results to bak directory (default=save)
                   (0=Current dir; -1=Not Saved,default)
        -f:   Save each SQL stmt to a different file in the $QUERIES dir
        -s:   Display N characters of SQL stmt (-1=all, default)
        -sql: Only display SQL statements, without statistics
        -i:   Include info in output, where Q could contain any of the following
               S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info
               N=None_of_the_above (default=SORTX)
        -p:   mon_name is a PIPE; Don't save the summary to an output file.
        -ps:  mon_name is a PIPE; Save the summary to an output file.
               Summary output is only visible when the event monitor closes.
        -q:   Quiet (default is to display output)
        -r:   Don't read the monitor files, reuse existing output (extraction)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of monitor output filename.
        Values for -s, -f, -sql and -i are part of Summary output filename.
        Pipes: Make sure to run this utility BEFORE starting the event monitor
               (prompts given).  For -ps only flushed summary output is
               visible in real-time (instructions given). So for real-time
               viewing, use -p, which doesn't save summary to a file.
        In most cases, best viewed in 132 column window.
```

The following is the source:

```
#!/bin/ksh
# mon_stmt.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
   echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname -m mon_name [-o N] [-f]
         [-s N] [-sql] [-i Q] [-p] [-ps] [-q] [-r] [-b]
Summerize STATEMENT statistics captured from an Event Monitor.
   -c:   Comment placed on first line of Summary output file (required)
   -d:   Database name (required)
   -m:   Monitor name (required)
   -o:   Save output in dir N under \$RESULTS
       (0=Current dir; -1=Not Saved,default)
   -f:   Save each SQL stmt to a different file in the \$QUERIES dir
   -s:   Display N characters of SQL stmt (-1=all, default)
   -sql: Only display SQL statements, without statistics
   -i:   Include info in output, where Q could contain any of the following
          S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info
          N=None_of_the_above (default=SORTX)
   -p:   mon_name is a PIPE; Don't save the summary to an output file.
   -ps:  mon_name is a PIPE; Save the summary to an output file.
          Summary output is only visible when the event monitor closes.
   -q:   Quiet (default is to display output)
   -r:   Don't read the monitor files, reuse existing output (extraction)
   -b:   DONT save prior (old) results to bak directory (default=save)
       (Not saved anyway unless -o is greater than 0)
   Notes:
   Value for -d is part of monitor output filename.
   Values for -s, -f, -sql and -i are part of Summary output filename.
   Pipes: Make sure to run this utility BEFORE starting the event monitor
          (prompts given).  For -ps only flushed summary output is
              visible in real-time (instructions given). So for real-time
          viewing, use -p, which doesn't save summary to a file.
   In most cases, best viewed in 132 column window.
"
}

# Constants

QUERIES=~/queries
RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`"
RES_EXT=".out"
SUM_EXT=".sum"
AWKSCRIPT="`dirname $0`/`basename $0 .ksh`.awk"

# Defaults

QUIET=0
DISPLAY_SQL=-1
SAVE_SQL=0
SQL_ONLY=0
RESULTS_DIR=-1# -1 defaults to not saved
REUSE_OUT=0
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

INCLUDE_OPT="SORTX"
PIPE=0
PIPE_SAVE=0
```

```
# Parse parameters

while [ "$1" != "" ]
do
   case "$1" in
   "-c") shift; COMMENT=$1; shift;;
   "-d") shift; DB_NAME=$1; shift;;
   "-m") shift; MON_NAME=$1; shift;;
   "-o") shift; RESULTS_DIR=$1; shift;;
   "-f") shift; SAVE_SQL=1;;
   "-s") shift; DISPLAY_SQL=$1; shift;;
   "-sql") shift; SQL_ONLY=1;;
   "-i") shift; INCLUDE_OPT=$1; shift;;
   "-p") shift; PIPE=1;;
   "-ps") shift; PIPE=1; PIPE_SAVE=1;;
   "-q") shift; QUIET=1;;
   "-r") shift; REUSE_OUT=1;;
   "-b") shift; SAVE_OLD_RESULTS=0;;
   *   ) shift; PARSE_ERROR="Invalid Param";;
   esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
   PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
   PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ "$MON_NAME" = "" ] && \
   PARSE_ERROR="${PARSE_ERROR} -Monitor Name is required"

[ $DISPLAY_SQL -ge -1 ] 2>/dev/null || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid number following -s param"

[ $RESULTS_DIR -ge -1 ] 2>/dev/null || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

echo "$PARAMS" | awk '/-sql/ && /-i/{exit -1}' || \
   PARSE_ERROR="${PARSE_ERROR} -Cant combine -sql with -i"

echo "$PARAMS" | awk '/-p[ $]/ && /-ps/{exit -1}' || \
   PARSE_ERROR="${PARSE_ERROR} -Cant combine -ps with -p"

echo "$PARAMS" | awk '/-p[ $]/ && /-q/{exit -1}' || \
   PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -p (only with -ps)"

echo "$PARAMS" | awk '/-o -1/ && /-r/{exit -1}' || \
   PARSE_ERROR="${PARSE_ERROR} -Cant combine -r with -o -1"

echo "$PARAMS" | awk '/-o -1/ && /-q/{exit -1}' || \
   PARSE_ERROR="${PARSE_ERROR} -Cant combine -q with -o -1"

[ $SQL_ONLY -eq 1 ] && INCLUDE_OPT="N"

echo "$INCLUDE_OPT" | awk '/[^SORTX]/ && ($0 != "N"){exit -1}' || \
   PARSE_ERROR="${PARSE_ERROR} -Invalid value for -i param"

if [ "$PARSE_ERROR" != "" ]
then
   echo ""
```

```
    echo $PARSE_ERROR
    echo ""
    display_syntax
    exit
fi

if [ $SAVE_SQL -eq 1 ]
then
    # Get last query file number
    mkdir $QUERIES 2>/dev/null
    LAST_QUERY=`ls $QUERIES | sort -n | tail -1`
    LAST_QUERY=`basename $LAST_QUERY .sql`
fi

RESULTS_FILE=`echo $MON_NAME | tr [a-z] [A-Z]`
DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}_${DISPLAY_SQL}${SAVE_SQL}${SQL_ONLY}${INCLUDE_OPT}${SUM_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
    RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
    RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
    SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
    RES_OUT=${RESULTS_FILE}${RES_EXT}
    SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $REUSE_OUT -eq 1 -a ! -f $RES_OUT ]
then
    echo "Can't reuse $RES_OUT - Missing"
    exit
fi

if [ $RESULTS_DIR -gt 0 ]
then
    mkdir $RESULTS 2>/dev/null
    mkdir $RESULTS/$RESULTS_DIR 2>/dev/null
    if [ $SAVE_OLD_RESULTS -eq 1 ]
    then
        mkdir $RESULTS/$RESULTS_DIR/bak2>/dev/null
        [ $REUSE_OUT -eq 0 ] && \
        cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
        cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
    fi
fi

TMP_SQL="${QUERIES}/sql.$$.tmp"

# Clean up previous aborts (trap didn't work :(
rm ${QUERIES}/sql.[0-9]*.tmp 2>/dev/null

export QUERIES DISPLAY_SQL SAVE_SQL SQL_ONLY LAST_QUERY TMP_SQL
export INCLUDE_OPT

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null
```

```
if [ $RESULTS_DIR -ge 0 ]
then
    R_OUTPUT="| tee $RES_OUT"

    # with -p, Summary output not saved
    if [ $PIPE -eq 1 -a $PIPE_SAVE -eq 0 ]
    then
        Q_OUTPUT=""
        echo "[Summary Output Not Saved]"
    else
        echo "[Creating: $SUM_OUT]"
    fi
else
    R_OUTPUT=""
    Q_OUTPUT=""
    echo "[No Output Saved]"
fi

eval echo "-- $COMMENT"$Q_OUTPUT
eval echo "-- ---------------------------------------"$Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"$Q_OUTPUT
eval echo "--             `date`"$Q_OUTPUT
eval echo "-- "          $Q_OUTPUT

if [ $PIPE -eq 0 -o $REUSE_OUT -eq 1 ]
then
    if [ $RESULTS_DIR -eq -1 ]
    then
        echo db2evmon $DB_NAME $MON_NAME
        echo ""
        db2evmon $DB_NAME $MON_NAME | awk -f $AWKSCRIPT
    else
        if [ $REUSE_OUT -eq 0 ]
        then
            eval echo db2evmon $DB_NAME $MON_NAME$Q_OUTPUT
            echo ""
            echo "[Creating: $RES_OUT]"
            db2evmon $DB_NAME $MON_NAME > $RES_OUT
        else
            echo ""
            echo "[Reusing: $RES_OUT]"
        fi

        echo ""
        eval awk -f $AWKSCRIPT $RES_OUT $Q_OUTPUT
    fi
else
    echo "\
--------------------------------------------------------------------
Start the event monitor to begin summarizing (filtering) as follows:
  Use: db2 SET EVENT MONITOR $MON_NAME STATE 1
--------------------------------------------------------------------"
#  or: . `dirname $0`/mon_state.ksh $MON_NAME 1
    if [ $PIPE_SAVE -eq 0 ]
    then
        # Gives better real-time display without pipe after awk
        echo ""
        echo db2evmon $DB_NAME $MON_NAME
        echo ""
        eval db2evmon $DB_NAME $MON_NAME $R_OUTPUT | awk -f $AWKSCRIPT
    else
        echo "\
Only when all pipe buffers are flushed will be output be visible.
```

```
To flush event monitor records manually
  Use: db2 FLUSH EVENT MONITOR $MON_NAME BUFFER
Closing the Event Monitor also flushes all records.
  Use: db2 SET EVENT MONITOR $MON_NAME STATE 0
----------------------------------------------------------------------"
#  or: . `dirname $0`/mon_state.ksh $MON_NAME 0
       eval echo ""      $Q_OUTPUT
       eval echo db2evmon $DB_NAME $MON_NAME $Q_OUTPUT
       eval echo ""      $Q_OUTPUT
       [ "$R_OUTPUT" != "" ] && echo "[Creating: $RES_OUT]"
       eval db2evmon $DB_NAME $MON_NAME $R_OUTPUT | \
           eval awk -f $AWKSCRIPT$Q_OUTPUT
   fi
fi
```

The mon_stmt.ksh script uses the mon_stmt.awk file as the following source:

```
BEGIN{
   OS          = "AIX";
   QUERIES    = ENVIRON[ "QUERIES"     ];
   DISPLAY_SQL = ENVIRON[ "DISPLAY_SQL" ];
   SAVE_SQL    = ENVIRON[ "SAVE_SQL"    ];
   SQL_ONLY    = ENVIRON[ "SQL_ONLY"    ];
   LAST_QUERY  = ENVIRON[ "LAST_QUERY"  ];
   TMP_SQL     = ENVIRON[ "TMP_SQL"     ];
   INCLUDE_OPT = ENVIRON[ "INCLUDE_OPT" ];

#       S=Start-Time ; O=Operation ; T=Timing ; R=Row-Counts; X=Sort-Info;N=None
   if (INCLUDE_OPT != "N")
   {
       if (INCLUDE_OPT ~ /S/) OPT_ST_TIME = 1;
       if (INCLUDE_OPT ~ /O/) OPT_OPER    = 1;
       if (INCLUDE_OPT ~ /T/) OPT_TIMES   = 1;
       if (INCLUDE_OPT ~ /R/) OPT_ROWS    = 1;
       if (INCLUDE_OPT ~ /X/) OPT_SORTS   = 1;
   }

   STMT_NUM    = 0;
   MIL         = 1000000;
   CON_REC     = 0;
   header      = 0;
   h=0;  # Counter of Handles
}

/) Connection Header Event .../{ CON_REC=1;            }
/  Appl Handle: / && CON_REC{ HAND[      ++h] = $NF ; H=$NF ;
                    CA_HAND[    H ] = $NF}
/  Appl Id: /&& CON_REC{ CA_ID[      H ] = $NF}
/  Appl Seq number: /&& CON_REC{ CA_SEQ[     H ] = $NF}
/  Program Name    : /{ C_PROG[     H ] = $NF}
/  Authorization Id: /{ C_DBUSER[   H ] = $NF}
/  Execution Id    : /{ C_CLUSER[   H ] = $NF}
/  Client Process Id: /{ C_CLPROC[   H ] = $NF}
/  Client Database Alias: /{ C_DBNAME[   H ] = $NF}
/  Client Communication Protocol: /{ C_PROTOCOL[ H ] = $NF}
/  Client Network Name: /{ C_NET[      H ] = $NF}
/  Connect timestamp: /{ C_TIME[     H ] = $(NF-1) $NF }
####################################################################
/) Statement Event .../{ CON_REC       = 0       ;
                    A_HANDLE= ""  ;
                                      A_ID= ""  ;
                                      A_SEQ= ""  ;
                                      S_TYPE= ""  ;
```

```
                                        S_OPER= ""  ;
                                        P_SECTION= ""  ;
                                        P_CREATOR= ""  ;
                                        P_PACKAGE= ""  ;
                                        S_CURSOR= ""  ;
                                        S_BLOCKED= ""  ;
                                        S_TEXT = ""  ;
                                        S_TYPE = ""  ;
                                        S_START_T= ""  ;
                                        S_EXEC_T= ""  ;
                                        S_AGENTS= ""      ;
                                        S_U_CPU_T= ""      ;
                                        S_S_CPU_T= ""      ;
                                        S_FETCHES= ""      ;
                                        S_SORTS= ""      ;
                                        S_SORT_T= ""      ;
                                        S_SORT_OFLOWS = ""      ;
                                        S_ROWS_READ= ""      ;
                                        S_ROWS_WRITTEN= ""      ;
                                        S_SQLCODE= ""      ;
}
#) Statement Event ...
/  Appl Handle: /&& ! CON_REC{ A_HANDLE= $NF  }
/  Appl Id: /&& ! CON_REC{ A_ID = substr( $NF, length($NF)-4) }
/  Appl Seq number: /&& ! CON_REC{ A_SEQ= $NF  }
#  -----------------------------------------
/  Type     : /{ S_TYPE= $NF  }
/  Operation: /{ S_OPER = substr( $0, 14)}
/  Section  : /{ P_SECTION= $NF  }
/  Creator  : /{ P_CREATOR= $NF  }
/  Package  : /{ P_PACKAGE= $NF  }
/  Cursor   : /{ S_CURSOR= $NF  }
/  Cursor was blocking: /{ S_BLOCKED= $NF  }
/  Text     : /{ S_TEXT = substr( $0, 14)}
#  -----------------------------------------
/  Operation: Static Commit/{ S_TYPE = "Static";
                  S_OPER= "Commit"}
/  Operation: Execute Immediate/{ S_OPER= "ExecImmed"}
#  -----------------------------------------
/  Start Time: /{ S_START_T= $NF  }
#  Stop Time:  05-16-2000 18:25:00.650866 - for datetime: $(NF-1) $NF }
/  Exec Time: /{ S_EXEC_T= $(NF-1) } #Sec
/  Number of Agents created: /{ S_AGENTS= $NF     }
/  User CPU: /{ S_U_CPU_T= $(NF-1) } #Sec
/  System CPU: /{ S_S_CPU_T= $(NF-1) } #Sec
/  Fetch Count: /{ S_FETCHES= $NF  }
/  Sorts: /    { S_SORTS= $NF  }
/  Total sort time: /{ S_SORT_T= $NF  } #Msec
/  Sort overflows: /{ S_SORT_OFLOWS = $NF  }
/  Rows read: /{ S_ROWS_READ= $NF  }
/  Rows written: /{ S_ROWS_WRITTEN= $NF  }
#  Internal rows deleted: 0
#  Internal rows updated: 0
#  Internal rows inserted: 0
#  SQLCA:
/   sqlcode: /{ S_SQLCODE= $NF  }
#   sqlstate: 00000

/   sqlstate: /{  # Begin Display

    STMT_NUM=STMT_NUM + 1;
    SQL_OUT="";
    if (DISPLAY_SQL == -1)
        SQL_LEN = length( S_TEXT);
    else
        SQL_LEN = DISPLAY_SQL;

    # Save SQL Text to a file, or get name of existing (duplicate) file
    if ( S_TEXT && SAVE_SQL )  Save_Sql();

    headline = "";
    dataline = "";

    if ( SQL_ONLY )
    {
        if (SAVE_SQL)S_Save_Info();
    }
    else
    {
        S_App_Info();
        if (SAVE_SQL)S_Save_Info();
        if (OPT_OPER) S_Oper_Info();
        if (OPT_TIMES) S_Timing_Info();
        if (OPT_ROWS)  S_Rows_Info();

        if (OPT_SORTS)
        {
            # When all options displayed, remove '|' to fit in 132
            if (INCLUDE_OPT ~ /[SORTX]{5}/)
            {
                rm_Trailing();
                headline = headline " ";
                dataline = dataline " ";
            }
            S_Sort_Info();
        }
    }

    S_SQL_Text();

    rm_Trailing();

    if ( headline && ! header)
    {
        for (i=1; i<=length( headline); i++)
            underline = underline "-";

        print headline;
        print underline;
        header = 1;
    }

    if (dataline) print dataline;
}

################################################################################
function S_App_Info() {

    headline = headline sprintf( \
            "%4.4s %5.5s %-4.4s",
            "Hand",
            "AppID",
            "Seq");
    dataline = dataline sprintf( \
            "%4s %5s %4s",
            A_HANDLE,
            A_ID,
            A_SEQ);

    if (OPT_ST_TIME)
    {
    headline = headline sprintf( \
            " %-15.15s",
            "Start-Time");
    dataline = dataline sprintf( \
            " %15s",
            S_START_T);
    }

    headline = headline " | ";
    dataline = dataline " | ";
}

################################################################################
function S_Save_Info() {

    headline = headline sprintf( \
            "%8.8s | ",
            "SQL-File");
    if ( ( ! SQL_ONLY ) || ( SQL_ONLY && S_TEXT ) )
    dataline = dataline sprintf( \
            "%8s | ",
            SQL_OUT);
}

################################################################################
function  S_Oper_Info() {

    headline = headline sprintf( \
            "%-2.2s %-9.9s %5.5s | ",
            "Type",
            "Oper",
            "Code");
    dataline = dataline sprintf( \
            "%-2.2s %-9s %5s | ",
            S_TYPE,
            S_OPER,
            S_SQLCODE );
}

################################################################################
function S_Timing_Info() {

    headline = headline sprintf( \
            "%8.8s %8.8s %8.8s | ",
            "Exec(s)",
            "UCPU(s)",
            "SCPU(s)");
    dataline = dataline sprintf( \
            "%8s %8s %8s | ",
            substr( S_EXEC_T, 1, 8),
            substr( S_U_CPU_T, 1, 8),
            substr( S_S_CPU_T, 1, 8));
}

################################################################################
function S_Rows_Info() {
    headline = headline sprintf( \
            "%9.9s %9.9s %5.5s | ",
            "Read",
            "Written",
            "Fetch");
    dataline = dataline sprintf( \
            "%9s %9s %5s | ",
            S_ROWS_READ,
            S_ROWS_WRITTEN,
            S_FETCHES);
}

################################################################################
function S_Sort_Info() {

    headline = headline sprintf( \
            "%5.5s %5.5s %9.9s | ",
            "Sorts",
            "SOVFL",
            "Sort(ms)");
    dataline = dataline sprintf( \
            "%5s %5s %9s | ",
            S_SORTS,
            S_SORT_OFLOWS,
            S_SORT_T);
}

################################################################################
function S_SQL_Text() {

    if ( DISPLAY_SQL )
        headline = headline sprintf( \
            "%-*s | ",
            SQL_LEN, "SQL-Text");
    if (S_TEXT && SQL_LEN)
        dataline = dataline sprintf( \
            "%-*.*s | ",
            SQL_LEN, SQL_LEN, S_TEXT);

    # exception to rm_Trailing, force removal of seperator when no SQL text
    # since this is always the last entry on the line
    if (S_TEXT == "")
        dataline = substr( dataline, 1, (len - 3) );
}

################################################################################
function rm_Trailing() {
# Get rid of trailing separator

    if ( substr( headline, (len=length( headline))-2) == " | " )
    {
        headline = substr( headline, 1, (len - 3) );
        dataline = substr( dataline, 1, (len - 3) );
    }
}

################################################################################
function Save_Sql() {
# Save SQL Text to a file, or get name of existing (duplicate) file

    DUPFILE="";
    print S_TEXT > TMP_SQL;
    close( TMP_SQL);
```

```
        # Check if a duplicate SQL file exists
        if ( OS == "AIX" )
        {
            # Get filesize of TMP_SQL
            "ls -l " TMP_SQL | getline DIRLINE;
            close( "ls -l " TMP_SQL );
            DFN=split( DIRLINE, DIRARRAY);
            FILESIZE=DIRARRAY[5];

            # Loop thru sql files in reverse creation order
            # Compare only if same filesize
            while ( (DUPFILE == "") &&
                ("ls -lt " QUERIES "/[0-9]*.sql" | getline DIRLINE ) )
            {
                DFN=split( DIRLINE, DIRARRAY);
                if ((DIRARRAY[5] == FILESIZE) &&
                    (TMP_SQL != DIRARRAY[DFN])  )
                {
                    "diff " TMP_SQL " " DIRARRAY[DFN]    \
                    " >/dev/null 2>&1 && "               \
                    "   basename "  DIRARRAY[DFN] | \
                    getline DUPFILE;
                close(  "diff " TMP_SQL " " DIRARRAY[DFN]    \
                    " >/dev/null 2>&1 && "               \
                    "   basename "  DIRARRAY[DFN] );
                }
            }
            close( "ls -lt " QUERIES "/[0-9]*.sql" );

            system( "rm " TMP_SQL);
        }
        else# Untested
        {
            # delete TMP_SQL
        }

        if ( DUPFILE != "" )
        {  # A duplicate SQL file was found
            SQL_OUT = DUPFILE;
        }
        else
        {  # Create SQL file
            LAST_QUERY=LAST_QUERY + 1;
            SQL_OUT=LAST_QUERY ".sql";
            print S_TEXT > QUERIES "/" SQL_OUT;
        }
    }
```

A.7 Benchmark tool

The `db2batch.ksh` shell script executes the `db2batch` tool and stores the output to the directory structure. The syntax of this script is as follows:

```
db2bench.ksh -c "comment" -d dbname [-u user/pwd] -q queryNum -o N
             [-ir rows] [-if rows] [-e m] [-v] [-b]
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -u:   Userid and password seperated by slash (default=db2inst1/db2inst1)
        -q:   Query number to execute. File with .sql extension must exist in
                $QUERIES directory (required)
        -o:   Save output in dir N under $RESULTS (required)
                (0=Current dir; -1=Not Valid; no default)
        -ir:  Rows to return to output file (-1=all,default=10)
        -if:  Rows to fetch even if not returned (-1=all,default)
        -e:   Explain level: 0=ExecuteOnly; 1=ExplainOnly; 2=Explain&Execute
                (default=2)
        -v:   Verbose (default is NOT to display output)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of output and summary filename.
```

The source is as follows:

```
#!/bin/ksh
# db2bench.ksh
# Raanon Reutlinger, IBM Israel, May 2000

display_syntax()
{
        echo "\
SYNTAX: `basename $0` -c \"comment\" -d dbname [-u user/pwd] -q queryNum -o N
              [-ir rows] [-if rows] [-e m] [-v] [-b]
Execute db2batch tool which executes the SQL found in a file and reports on
execution times and snapshot information.
        -c:   Comment placed on first line of Summary output file (required)
        -d:   Database name (required)
        -u:   Userid and password seperated by slash (default=db2inst1/db2inst1)
        -q:   Query number to execute. File with .sql extension must exist in
                \$QUERIES directory (required)
        -o:   Save output in dir N under \$RESULTS (required)
                (0=Current dir; -1=Not Valid; no default)
        -ir:  Rows to return to output file (-1=all,default=10)
        -if:  Rows to fetch even if not returned (-1=all,default)
        -e:   Explain level: 0=ExecuteOnly; 1=ExplainOnly; 2=Explain&Execute
                (default=2)
        -v:   Verbose (default is NOT to display output)
        -b:   DONT save prior (old) results to bak directory (default=save)
                (Not saved anyway unless -o is greater than 0)
        Notes:
        Value for -d is part of output and summary filename.
"
}

# Constants
```

```
QUERIES=~/queries
RESULTS=~/results
RESULTS_FILE="`basename $0 .ksh`.awk"
RES_EXT=".out"
SUM_EXT=".sum"

# Defaults

QUIET=1
RESULTS_DIR=""          # no default
SAVE_OLD_RESULTS=1
PARSE_ERROR=""
PARAMS=$*

USERPASS="db2inst1/db2inst1"
QRYNUM=""
IROWS=10
IFETCH=-1
EXPLAIN=2

# Parse parameters

while [ "$1" != "" ]
do
        case "$1" in
        "-c"    ) shift; COMMENT=$1;     shift  ;;
        "-d"    ) shift; DB_NAME=$1;     shift  ;;
        "-u"    ) shift; USERPASS=$1;    shift  ;;
        "-q"    ) shift; QRYNUM=$1;      shift  ;;
        "-ir"   ) shift; IROWS=$1;       shift  ;;
        "-if"   ) shift; IFETCH=$1;      shift  ;;
        "-e"    ) shift; EXPLAIN=$1;     shift  ;;
        "-o"    ) shift; RESULTS_DIR=$1; shift  ;;
        "-v"    ) shift; QUIET=0                ;;
        "-b"    ) shift; SAVE_OLD_RESULTS=0     ;;
        *       ) shift; PARSE_ERROR="Invalid Param"    ;;
        esac
done

# Verify parameters

[ "$COMMENT" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Comment is required"

[ "$DB_NAME" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Database name is required"

[ "$QRYNUM" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Query number is required"  || \
[ $QRYNUM -ge 0 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -q param"

[ "$RESULTS_DIR" = "" ] && \
        PARSE_ERROR="${PARSE_ERROR} -Output directory number is required" || \
[ $RESULTS_DIR -ge 0 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -o param"

echo "$USERPASS" | awk -F/ '($1 == "") || ($2 == ""){exit -1}' || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid user/pass following -u"
[ $IROWS -ge -1 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -ir param"
```

```
[ $IFETCH -ge -1 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -if param"

[ $EXPLAIN -ge 0 -a $EXPLAIN -le 2 ] 2>/dev/null || \
        PARSE_ERROR="${PARSE_ERROR} -Invalid number following -e param"

if [ "$PARSE_ERROR" != "" ]
then
        echo ""
        echo $PARSE_ERROR
        echo ""
        display_syntax
        exit
fi

DB_NAME=`echo $DB_NAME | tr [a-z] [A-Z]`
RESULTS_FILE=$QRYNUM

RES_EXT="_${DB_NAME}${RES_EXT}"
SUM_EXT="_${DB_NAME}${SUM_EXT}"

if [ $RESULTS_DIR -gt 0 ]
then
        RES_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${RES_EXT}
        SUM_OUT=${RESULTS}/${RESULTS_DIR}/${RESULTS_FILE}${SUM_EXT}
        RES_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${RES_EXT}.$$
        SUM_BAK=${RESULTS}/${RESULTS_DIR}/bak/${RESULTS_FILE}${SUM_EXT}.$$
else
        RES_OUT=${RESULTS_FILE}${RES_EXT}
        SUM_OUT=${RESULTS_FILE}${SUM_EXT}
fi

if [ $RESULTS_DIR -gt 0 ]
then
        mkdir $RESULTS                  2>/dev/null
        mkdir $RESULTS/$RESULTS_DIR     2>/dev/null
        if [ $SAVE_OLD_RESULTS -eq 1 ]
        then
                mkdir $RESULTS/$RESULTS_DIR/bak 2>/dev/null
                cp $RES_OUT $RES_BAK 2>/dev/null && echo "[Created: $RES_BAK]"
                cp $SUM_OUT $SUM_BAK 2>/dev/null && echo "[Created: $SUM_BAK]"
                [ -f $RES_BAK ] && \
                        echo "[Modifying: $RES_BAK]" && \
                        `dirname $0`/db2bench_prefix.ksh $RES_BAK

        fi
fi

[ $QUIET -eq 1 ] && VERBOSE="off" || VERBOSE="on"

# BEGIN .......

[ $QUIET -eq 1 ] && Q_OUTPUT=">> $SUM_OUT" || Q_OUTPUT="| tee -a $SUM_OUT"
rm $SUM_OUT 2>/dev/null

echo "[Creating: $SUM_OUT]"
echo "[Creating: $RES_OUT]"
echo ""

eval echo "-- $COMMENT"                                $Q_OUTPUT
eval echo "-- ----------------------------------------" $Q_OUTPUT
eval echo "-- Invocation: $0 $PARAMS"                  $Q_OUTPUT
eval echo "--             `date`"                      $Q_OUTPUT
eval echo "-- "                                         $Q_OUTPUT

echo "-- $COMMENT"                                      >  $RES_OUT
echo "-- ---------------------------------------"      >> $RES_OUT
echo "-- Invocation: $0 $PARAMS"                        >> $RES_OUT
echo "--             `date`"                            >> $RES_OUT
echo "-- "                                              >> $RES_OUT

eval echo db2batch \
        -d $DB_NAME \
        -f $QUERIES/${QRYNUM}.sql \
        -r ${RES_OUT},${SUM_OUT} \
        -a db2inst1/db2inst1 \
        -c off \
        -i complete \
        -o r $IROWS  f $IFETCH  p 5  e $EXPLAIN \
        -v $VERBOSE                                     $Q_OUTPUT
eval echo ""                                            $Q_OUTPUT

echo db2batch \
        -d $DB_NAME \
        -f $QUERIES/${QRYNUM}.sql \
        -r ${RES_OUT},${SUM_OUT} \
        -a db2inst1/db2inst1 \
        -c off \
        -i complete \
        -o r $IROWS  f $IFETCH  p 5  e $EXPLAIN \
        -v $VERBOSE                                     >> $RES_OUT
echo ""                                                 >> $RES_OUT

db2batch \
        -d $DB_NAME \
        -f $QUERIES/${QRYNUM}.sql \
        -r ${RES_OUT}.tmp,${SUM_OUT}.tmp \
        -a db2inst1/db2inst1 \
        -c off \
        -i complete \
        -o r $IROWS  f $IFETCH  p 5  e $EXPLAIN \
        -v $VERBOSE

[ ! -f ${RES_OUT}.tmp ] && echo "Error! File not created!
        RES_OUT=${RES_OUT}.tmp" &&
        exit

cat ${RES_OUT}.tmp                                      >> $RES_OUT && \
rm  ${RES_OUT}.tmp

cat ${SUM_OUT}.tmp                                      >> $SUM_OUT && \
rm  ${SUM_OUT}.tmp

# Prefix selected lines with filename
echo ""
echo "[Modifying: $RES_OUT]"
`dirname $0`/db2bench_prefix.ksh $RES_OUT
#db2batch -d tpc -f queries/2.sql -a db2inst1/db2inst1 -c off -i complete -o r 2
 f -1 p 5 e 0 -v off | awk '/Summary of Results/{doprint=1}doprint||/seconds *$/
{print}'
```

The db2bench.ksh script calls the following db2bench_prefix.ksh file:

```
#!/bin/ksh
# db2bench_prefix.ksh
# Raanon Reutlinger, IBM Israel

[ "$1" = "" ] && echo "\
SYNTAX: `basename $0` {db2batch-Output-Filename} [Override-Prefix]
Prefixes the filename on selected lines from the db2batch-Output file.

" && exit

SOURCE1=$1
SOURCE=`basename $0`.$$.source
TARGET=`basename $0`.$$.tmp

PREFIX=${2:-`basename $SOURCE1`}
export PREFIX

[ ! -f $SOURCE1 ] && echo "$SOURCE1 doesn't exist" && exit

# To fix problem of missing new-line at end of $SOURCE1 files
awk '{print}' $SOURCE1 > $SOURCE

awk -f `dirname $0`/db2bench_prefix.awk $SOURCE > $TARGET

[ ! -f $SOURCE -o ! -f $TARGET ] && echo "Error! File not created!
        SOURCE=$SOURCE
        TARGET=$TARGET" &&
        exit

LINES_ORIG=`wc -l $SOURCE | awk '{print $1}'`
LINES_TMP=` wc -l $TARGET | awk '{print $1}'`

# Only copy over SOURCE1 if number of lines match exactly
[ $LINES_ORIG -eq $LINES_TMP ]  && mv $TARGET $SOURCE1 \
                                || echo "Error: file not modified"

# As long as SOURCE1 is still around delete TARGET and SOURCE(tmp) if mv failed
[ -f $SOURCE1 ] && rm $TARGET $SOURCE 2>/dev/null
```

The `db2bench.ksh` script calls the following `db2bench_prefix.awk` file. You need to have it in the same directory.

```
BEGIN{
        TOP        = 1;
        PREFIX     = ENVIRON[ "PREFIX" ];
}
{
        PUT_PREFIX = 0 ;
}
#/Statement number: /                                  { TOP=0 }
#TOP                                                   { PUT_PREFIX=1 }
(NR <= TOP)                                            { PUT_PREFIX=1 }
/           \*\*\* Database Snapshot \*\*\*/           { PUT_PREFIX=1 }
/           \*\*\* Bufferpool Snapshot \*\*\*/         { PUT_PREFIX=1 }
/Bufferpool Name  .* = /                               { PUT_PREFIX=1 }
/           \*\*\* Bufferpool Info Snapshot \*\*\*/    { PUT_PREFIX=1 }
/           \*\*\* Application Info Snapshot \*\*\*/   { PUT_PREFIX=1 }
/           \*\*\* Application Snapshot \*\*\*/        { PUT_PREFIX=1 }
/           \*\*\* Statement Details  \*\*\*/          { PUT_PREFIX=1 }
/Statement operation  .* = /                           { PUT_PREFIX=1 }
/       \*\*\* List of Associated Agents \*\*\*/       { PUT_PREFIX=1 }
/           \*\*\* Tablespace Header Snapshot \*\*\*/  { PUT_PREFIX=1 }
/           \*\*\* Tablespace Snapshot \*\*\*/         { PUT_PREFIX=1 }
/Tablespace Name .* = /                                { PUT_PREFIX=1 }
/           \*\*\* Table Header Snapshot \*\*\*/       { PUT_PREFIX=1 }
/           \*\*\* Table Snapshot \*\*\*/              { PUT_PREFIX=1 }
/Table Name .* = /                                     { PUT_PREFIX=1;TAB=$NF}
```

```
/Table File ID .* = / && (TAB == "=")                   { PUT_PREFIX=1 }
/           \*\*\* Database manager Snapshot \*\*\*/   { PUT_PREFIX=1 }
/           \*\*\* FCM Snapshot \*\*\*/                { PUT_PREFIX=1 }

{               sub( /^\[.*\]: /, "")   }       # Strip existing prefix
PUT_PREFIX{     printf "[%-15.15s]: ", PREFIX } # Prefix
{               print                   }
```

Appendix B. Using the additional material

This redbook also contains additional material in diskette format. See the appropriate section below for instructions on using or downloading each type of material.

B.1 Using the diskette

The diskette that accompanies this redbook contains the following:

File name	*Description*
db2bench.ksh	Executing db2batch command
db2bench_prefix.awk	Executing db2batch command
db2bench_prefix.ksh	Executing db2batch command
db2look.ksh	Executing db2look command
iostat.ksh	Executing iostat command
iostat.awk	Executing iostat command
list_proc.ksh	Printing application and process information
locks.ksh	Display lock information
locks.awk	Display lock information
mon_stmt.ksh	Report SQL statements information
mon_stmt.awk	Report SQL statements information
sqlcache.ksh	Display statements in the dynamic SQL cache
sqlcache.awk	Display statements in the dynamic SQL cache
upd_cfg.ksh	Executing GET DBM CFG or GET DB CFG
upd_cfg.awk	Executing GET DBM CFG or GET DB CFG

B.1.1 System requirements for using the diskette

You can use the sample scripts in the diskette on any UNIX operating systems on which the korn shell and awk are available.

B.1.2 How to use the diskette

As shown in Table 9 on page 120, create a directory “`work`” under your home directory and copy the contents of the diskette into this directory. The syntax to execute each script is shown in Appendix A, “Sample scripts” on page 335.

Before using the scripts, execute the following command to make them executable:

```
chmod +x *.ksh
```

B.2 Locating the additional material on the Internet

The diskette, or Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG246012`

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

Appendix C. Special notices

This publication is intended to help system designers, system administrators, and database administrators to set up, use, tune and maintain DB2 Universal Database Version 7.1 for optimal performance. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 Universal Database Version 7.1. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 Universal Database Version 7.1 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee

e (logo)®	Redbooks
IBM ®	Redbooks Logo
AIX	AS/400
AT	CT
Current	DataJoiner
DB2	DB2 Connect
DB2 Universal Database	DRDA
Enterprise Storage Server	Manage. Anything. Anywhere.
Netfinity	OS/2
RETAIN	RS/6000
SP	System/390
Wizard	XT
Lotus	400
Notes	Tivoli
TME	NetView
Cross-Site	Tivoli Ready
Tivoli Certified	

Appendix D. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 389.

- *Database Performance on AIX in DB2 UDB and Oracle Environments,* SG24-5511
- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810
- *RS/6000 Performance Tools in Focus*, SG24-4989

D.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at **ibm.com**/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

D.3 Other resources

These publications are also relevant as further information sources:

- *DB2 UDB for UNIX, Quick Beginnings,* GC09-2970
- *DB2 UDB Installation and Configuration Supplement*, GC09-2957
- *DB2 UDB SQL Reference, Volume 1*, SC09-2974
- *DB2 UDB SQL Reference, Volume 2*, SC09-2975

- *DB2 UDB System Monitor Guide and Reference*, SC09-2956
- *DB2 UDB Command Reference*, SC09-2951
- *DB2 UDB Data Movement Utilities Guide*, SC09-2955
- *DB2 UDB Application Building Guide*, SC09-2948
- *DB2 UDB Application Development Guide*, SC09-2949
- *DB2 UDB Call Level Interface Guide and Reference*, SC09-2950
- *DB2 UDB Administration Guide: Planning,* SC09-2946
- *DB2 UDB Administration Guide: Implementation,* SC09-2944
- *DB2 UDB Administration Guide: Performance*, SC09-2945
- *DB2 UDB Administrative API Reference*, SC09-2947

All DB2 UDB manuals are available at the DB2 Product and Service Technical Library. Visit the following URL:

`http://www-4.ibm.com/software/data/db2/library`

These publications are relevant as information sources for the performance tuning on AIX:

- *AIX Commands Reference*, SBOF-1851
- *AIX Performance Tuning Guide by Frank Waters*, ISBN 0-13-386707-2
- *Accelerating AIX: Performance Tuning for Programmers and System Administrators by Rudy Chukran*, ISBN 0-20-163382-5

AIX standard documentation can be found at:

`http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/`

These publications are relevant as information sources for performance tuning on Sun Solaris:

- *Sun Performance and Tuning: Java and the Internet by Adrian Cockcroft, Richard Pettit, Sun Microsystems*, ISBN 0-13-095249-4
- *Solaris Performance Administration: Performance Measurement, Fine Tuning, and Capacity Planning for Releases 2.5.1 and 2.6 by H. Frank Cervone*, ISBN 0-07-011768-3

D.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- `http://www-4.ibm.com/software/data/`
  IBM Data Management Home Page
- `http://www-4.ibm.com/software/data/db2/udb/winos2unix/support/`
  DB2 Universal Database and DB2 Connect Online Support
- `http://www-4.ibm.com/software/data/db2/db2tech/indexsvc.html/`
  DB2 Maintenance - Fixpaks for DB2 UDB
- `http://www.sunworld.com/sunworldonline/common/cockcroft.letters.html/`
  Adrian Cockcroft's Performance Q&A column in SunWorld
- `http://docs.sun.com/`
  Sun Product Documentation
- `http://access1.sun.com/`
  Sun Software support and consulting
- `http://access1.sun.com/patch.recommended/rec.html/`
  Sun Software support service - Recommended patches for Solaris

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** **ibm.com**/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-6012-00 DB2 UDB V7.1 Performance Tuning Guide
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	 O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

IBM

Redbooks

DB2 UDB V7.1 Performance Tuning Guide

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

DB2 UDB V7.1 Performance Tuning Guide

A comprehensive guide to improving DB2 UDB database performance

Efficient disk layout, logical and physical database design

Many tips to improve database performance

DISKETTE INCLUDED

This IBM Redbook will provide you with guidelines for system design, database design, and application design with DB2 UDB for AIX Version 7.1. We will also discuss the methods that are available for performance analysis and tuning.

Prior to reading this book, you need to have some knowledge of database environments, as well as a basic understanding of activities that are typically performed in a database environment.

This book was written from the AIX operating system perspective, and some tools and commands discussed in this book may not be available on other operating systems. However, the hints and methodologies described in this book can be applicable for database systems which use DB2 UDB on other operating systems.

SG24-6012-00 ISBN 0738418595